Boas Práticas – Padrões de Projeto: Abstract Factory, Iterator e Observer

# Clube Delphi

Boa Ideia – NoSQL com Delphi XE2
Entenda o que é NoSQL e como utilizar

QuickUpdate: Sockets
Comunicação cliente/servidor fácil

Ano 12 :: Edição 138 :: R$ 11,90

DevMedia

# NF-e com Delphi XE2

## Aplicação completa no FireMonkey

Acessando WebServices
Formas de acesso e ainda uma aplicação robô

Windows Phone
Utilizando controles Silverlight em seu aplicativo móvel

ISSN 1517990-7

# Somos tão apaixonados por tecnologia que o nome da empresa

# diz tudo.

**Porta 80** é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

### Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

### Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

### Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

### 1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

# Sumário


Quick Update

**06 – Sockets: Protocolo de Comunicação**
[ *Rafael Isique Guardieri Brendolan e Wanderley Donizeti Panosso Junior* ]

Delphi

Delphi

**10 – Mini curso NF-e e Delphi XE 2**
[ *Giuliano Scombatti Pinto* ]

Easy

**18 – Acessando WebServices**
[ *Giuliano Scombatti Pinto* ]

Delphi, Novidades

**26 – Prism XE2, Windows Phone e Silverlight**
[ *Fabrício Hissao Kawata* ]

Boas Práticas

**35 – NoSQL com Delphi XE2**
[ *Paulo Quicoli* ]

Boas Práticas

Boas Práticas

**42 – Padrões de Projeto: Abstract Factory, Iterator e Observer**
[ *Renato Groffe* ]


Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

## Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
*www.devmedia.com.br/clubedelphi/feedback*


RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Clube Delphi

Ano 12 - 138ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

## Corpo Editorial

**Editor Geral**

Paulo Quicoli
pauloquicoli@devmedia.com.br

**Editor Técnico**

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

**Equipe Editorial**

Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Guinther Pauli
guintherpauli@gmail.com

**Jornalista Responsável**

Kaline Dolabella - JP24185

**Atendimento ao Leitor**

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

*www.devmedia.com.br/mancad*
(21) 3382-5038

**Publicidade**

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

**Distribuição**

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

**Fale com o Editor!**

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

**Paulo Quicoli - Editor da Revista**
*pauloquicoli@gmail.com*

## EDITORIAL

Hoje o desenvolvimento móvel não é mais uma tendência, é uma realidade. Esse mercado movimenta importâncias cada vez maiores e devemos estar atentos a isso. Grandes empresas não querem ver essa onda simplesmente passar, mas querem fazer parte dela. E nós, muitas vezes pequenos, podemos ter uma participação nisso. A própria Embarcadero enxergou isso e trouxe o mundo IOS e Android ao Desenvolvedor Delphi. Esses são os nomes que mais aparecem hoje no Brasil, contudo, a Microsoft está apostando forte em seu produto, o Windows Phone, e criou uma parceria com a Nokia. Talvez você ache que isto não é importante, e realmente, aqui no Brasil ainda não chama muito a atenção, mas no exterior, indo além de EUA, França, China, todos aguardam as grandes novidades da Nokia com o Windows Phone. Assim, você pode criar aplicativos para o MarketPlace internacional. Mas para começar, é preciso conhecer, então nosso colega Fabrício mostra como utilizar o Silverlight para desenvolvimento Windows Phone, usando Prism XE2.

Voltando para o nosso mercado, um assunto que não descansa é o NF-e. E para unir o "útil ao agradável", o Giuliano inicia um mini curso, criando um aplicativo completo com Firemonkey que irá cuidar de tudo sobre NF-e. Ele também apresenta um assunto correlacionado, o acesso a webservices, e até mesmo o desenvolvimento de um aplicativo robô que pode preencher automaticamente formulários web. Assim, se você não implementou nada em relação à Internet no Delphi, este pode ser seu primeiro contato.

Quem costuma sair um pouco do "círculo Delphi", tem notado uma nova sigla circulando, o NoSQL. Eu mesmo trago um artigo sobre o assunto, explicando o que é o NoSQL, quais são seus fundamentos, onde é melhor aplicado, onde não deve ser aplicado e temos a introdução de um banco bastante conhecido e utilizado no meio, o mongoDB.

Para encerrar, temos mais um artigo do Renato sobre padrões de projeto, agora sobre Abstract Factory, Iterator e Observer.

Boa leitura!

**Paulo Quicoli**
*pauloquicoli@gmail.com*
*Editor Geral*
*quicoli.wordpress.com*
*twitter.com/pauloquicoli*

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia:
**www.devmedia.com.br/mvp**

DevMedia

# Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

**Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.**

TODAS as vídeo aulas que você sempre quis ver a partir de

**R$ 1,20**

**ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!**

**Saiba mais sobre o Sistema de Créditos!**

**www.devmedia.com.br/creditos**

DevMedia

# Sockets: Protocolo de

Muitas vezes é necessário fazer com que aplicações independentes comuniquem-se entre si. Uma situação comum em que essa necessidade é encontrada é no caso dos CHATs. Essa comunicação pode ser realizada em Delphi através do uso de sockets.

Os Sockets funcionam como meios de comunicação onde é possível o envio e a recepção de informações de uma forma clara e rápida, através de Protocolo TCP/IP. Neste QuickUpdate mostraremos como utilizar esse recurso.

## Conceitos importantes

Os Sockets trabalham baseados em uma estrutura Client/Server, onde ambos podem estar em aplicações distintas. Seu funcionamento básico é o seguinte:

- Todo Client deve se conectar a um e apenas um Server para se comunicar com outros Clients;
- Um Client nunca conseguirá se conectar diretamente a outro Client, pois sempre precisará de um Server como intermediário;
- Todo Client só pode enviar informações para o Server onde está conectado;
- O Server é responsável em receber essas informações e encaminhá-las para o Client de destino;

## Conhecendo os componentes

Os componentes utilizados para a implementação de aplicações baseadas em Sockets já vem com o Delphi, porém não são instalados por padrão. Para instalar o pacote de componentes acesse o Menu Component > Install Packages... > Add... , e adicione o pacote dcl sockets70.bpl que se encontra no diretório Bin da instalação do Delphi. Depois clique em OK para que os componentes sejam adicionados à paleta de componentes chamada Internet do Delphi. Dois novos componentes são adicionados, o ServerSocket e o ClientSocket.

**Nota**

Apesar de este exemplo ter sido implementado em Delphi 7, o mesmo pode ser reproduzido no Delphi XE 2, inclusive o pacote de sockets também não é instalado por padrão no XE 2. Para esta versão o arquivo a ser adicionado é o dcsockets160.bpl

## ServerSocket

O componente ServerSocket (**Figura 1**) é responsável em prover o acesso aos Clients, uma propriedade importante desse componente é a Port, onde deve ser especificado o número da porta pela qual o Client deverá conectar-se ao ServerSocket.

**Figura 1.** ServerSocket

O ServerSocket é responsável em gerenciar os Clients que estão conectados a ele, isso é gerenciado através de uma estrutura de pilha simples.

Todo Client ao se conectar a um ServerSocket é adicionado a uma pilha de conexões onde este recebe um ID que inicia-se em zero e vai até o máximo permitido para um tipo Integer. Quando um Client conecta-se, ele recebe o último ID da pilha e quando um outro Client desconecta-se do ServerSocket, todos os Clients com um ID superior a ele são atualizados, sendo decrescida uma posição na pilha.

## ClientSocket

Já o componente ClientSocket (**Figura 2**) nada mais é que o outro lado da história, sua tarefa é conectar-se a um ServerSocket e enviar ou receber informações desse ServerSocket.

**Figura 2.** ClientSocket

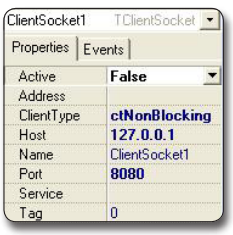



**Figura 3.** Propriedades do ClientSocket

Para que o ClientSocket se conecte a um ServerSocket, duas propriedades são indispensáveis, a propriedade Host onde deve-se informar o IP da máquina onde o ServerSocket estará sendo executado e a propriedade Port que especifica a porta em que o ServerSocket está trabalhando (**Figura 3**). Essa propriedade deve possuir o mesmo valor da propriedade Port especificada no ServerSocket.

## Aplicação prática

Para demonstrarmos o funcionamento dos Sockets, iremos implementar uma estrutura de CHAT simples que possua duas aplicações distintas comunicando-se via Socket.

# Comunicação

Rafael Isique Guardieri Brendolan e Wanderley Donizeti Panosso Junior

**Nota**

Para que não ocorra nenhum transtorno, é interessante que sejam utilizadas portas que não são utilizadas por outros aplicativos, existem 65.535 portas disponíveis para uso e várias destas portas não são utilizadas.

## Servidor

Inicialmente criaremos a aplicação do Servidor. Crie uma nova aplicação e deixe o formulário parecido com a **Figura 4.**

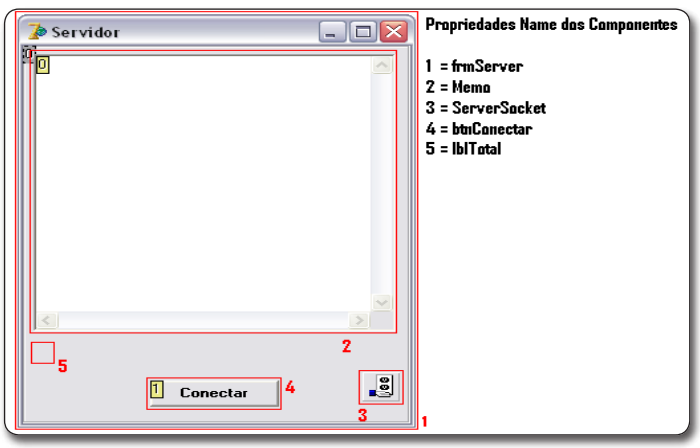



**Figura 4.** Aparência do formulário do servidor

O comportamento inicial do servidor será o seguinte: ao iniciar sua execução tentamos abrir a porta de comunicação, através da propriedade Active da classe ServerSocket. Contudo, ao finalizar o servidor é necessário fechar a porta de comunicação. Essa operação de abertura e fechamento foi encapsulado no procedimento ConectarDesconectar, que pode ser visto na **Listagem 1**. Desta forma, no evento OnCreate do formulário chamamos esse procedimento e no evento OnClose verificamos se a porta está aberta ou seja, se ServerSocket.Active é igual a true. Se estiver, chamamos o procedimento ConectarDesconectar para fechar a porta.

O servidor também deve informar o total de clientes conectados no serviço de chat e listar cada um deles. Isso é facilmente implementado porque o ServerSocket oferece o evento OnClient Conect que é disparado sempre que um novo ClientSocket se conecta. Assim, nesse evento atualizaremos a lblTotal com o total de clientes conectados ao Chat, além de adicionar ao Memo a nova conexão. O próprio ServerSocket já possui uma propriedade que registra a quantidade de conexões, nenhum artifício é necessário para obter essa contagem. Isso é visto na **Listagem 2**. O procedimento inverso, que é o de registrar a saída de um cliente, também é coberto pelo SocketServer através do evento OnClientDisconnect, que pode ser implementado de forma similar ao evento de conexão.

Nosso servidor não é um servidor que permite o que chamamos de "conversa privada" entre os clientes conectados, assim, quando um cliente envia uma mensagem, o servidor a repassa para todos. A **Listagem 3** exibe essa implementação.

**Listagem 1.** Implementação do Evento OnCreate do frmServer

```
procedure TfrmServer.ConectarDesconectar;
begin
ServerSocket.Active := not ServerSocket.Active;
if ServerSocket.Active then
  begin
    btnConectar.Caption := 'Desconectar';
    Memo.Lines.Add('Servidor Ligado!');
  end
  else
  begin
    btnConectar.Caption := 'Conectar';
    Memo.Lines.Add('Servidor Desligado!');
    lblTotal.Caption := '';
  end;
end;
```

**Listagem 2.** Registrando clientes que se conectam

```
procedure TfrmServer.ServerSocketClientConnect(Sender: TObject;
 Socket: TCustomWinSocket);
begin
  lblTotal.Caption := 'Clientes Conectados :' +
    IntToStr(ServerSocket.Socket.ActiveConnections);
  Memo.Lines.Add(Socket.LocalHost + ' conectou-se ao Chat!');
end;
```

**Listagem 3.** Distribuindo uma mensagem

```
procedure TfrmServer.ServerSocketClientRead(Sender: TObject;
 Socket: TCustomWinSocket);
var
  Texto : String;
  i : Integer;
begin
  Texto := Socket.ReceiveText;
  for i:= 0 to ServerSocket.Socket.ActiveConnections - 1 do
    ServerSocket.Socket.Connections[i].SendText(Texto);
end;
```

Sempre que uma nova mensagem é recebida pelo servidor, o evento OnClientRead é disparado. Assim recebemos o texto enviado por um cliente através da propriedade ReceiveText e

percorremos a propriedade Connections, que é uma lista que mantém uma referência a todos os clientes conectados e, para cada cliente contido, disparamos o método SendText repassando o texto originalmente recebido.

## Cliente

A aplicação cliente é um outro aplicativo, que deve ter ser formulário principal como mostra a **Figura 5.**

Um cliente se conecta a um servidor específico, identificado por seu endereço IP e porta de comunicação. Assim, de forma similar ao servidor, encapsulamos a tarefa de conexão e desconexão em um método chamado ConectarDesconectar (**Listagem 4**). Assim, ele é chamado no evento OnClick do botão Conectar.

Um cliente deve permitir o envio de mensagens, assim, no evento OnClick do botão btnEnviar enviamos um texto padrozinado. Nesse texto temos as Tags <NICK> e </NICK> onde é enviado o Nick do Usuário, e entre as Tags <MENSAGEM> e </MENSAGEM> é enviada a mensagem digitada pelo mesmo. A **Listagem 5** demonstra isso.

Podemos fazer a aplicação cliente se comportar de forma adequada à situação de conexão ou desconexão. O ClientSocket disponibiliza o evento OnConnect que é disparado sempre que o cliente se conecta a um servidor, assim, nesse evento podemos registrar o que for necessário. De forma semelhante temos o evento OnDisconnect, que é disparado quando o cliente se desconecta do servidor.

Como estamos lidando com conexão remota, estamos sucetíveis às mais diversas falhas e até o momento não realizamos nenhum tratamento direto. Para cobrir essa situação, o ClientSocket disponibiliza o evento OnError, que disponibiliza detalhes do acontecido. A **Listagem 6** exibe uma implementação bem simples do evento.

Nosso cliente até o momento se conecta, desconecta, envia mensagens e detecta erros. Ele também precisa lidar com as mensagens recebidas. Isso acontece no evento OnRead (**Listagem 7**). Como nossa mensagem vem codificada, foi criado um método que recebe como parâmetro o texto que o ClientSocket recebeu no evento OnRead, além de receber as TAGs inicial e final para que seja retornado o texto contido entre essas TAGs. Dessa forma é possível utilizar essa função genericamente para extrair qualquer informação baseada no protocolo em que está sendo desenvolvida a comunicação entre as partes. A **Listagem 8** mostra o método GetTag.

Após isso as duas aplicações já estão aptas a estabelecer uma comunicação. Primeiro inicie o Servidor e depois o Cliente. Para conectar-se ao Servidor basta adicionar no campo IP, o IP da máquina onde o Servidor esta sendo executado, e no campo NICK informe um nick de sua preferência.

**Listagem 4.** Conectando a um servidor

```
procedure TfrmCliente.ConectarDesconectar;
begin
  if not ClientSocket.Active then
    ClientSocket.Host := edtServer.Text;
  ClientSocket.Active := not ClientSocket.Active;
end;
```

**Listagem 5.** Enviando uma mensagem

```
procedure TfrmCliente.btnEnviarClick
(Sender: TObject);
begin
  if (ClientSocket.Active) then
    if (Trim(edtMsg.Text) <> '') then
ClientSocket.Socket.SendText('<NICK>' + edtNick.Text + '</NICK>' + '<MENSAGEM>' + edtMsg.Text + '</MENSAGEM>');
  edtMsg.Clear;
end;
```

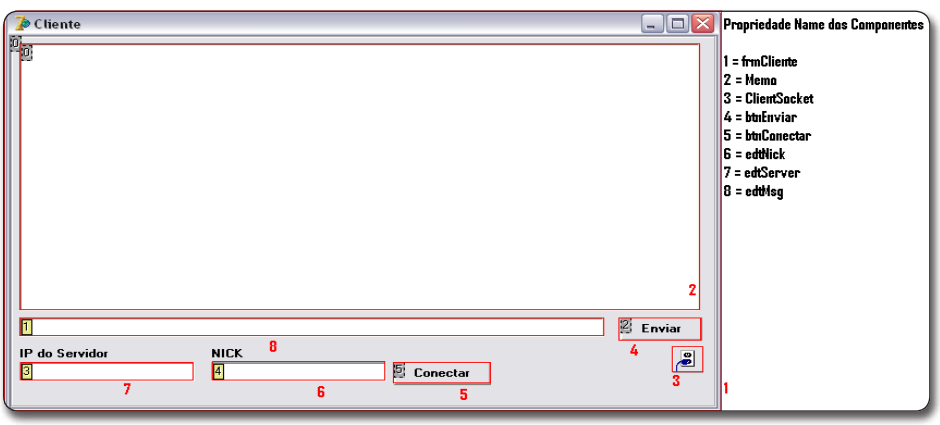



**Figura 5.** Aparência do Formulário do Cliente

**Listagem 6.** Identificando um erro de conexão

```
procedure TfrmCliente.ClientSocketError
(Sender: TObject;
 Socket: TCustomWinSocket; ErrorEvent:
 TErrorEvent;
 var ErrorCode: Integer);
begin
  MessageDlg('Servidor Não Encontrado!',
  mtError,mbOKCancel,0);
  ErrorCode := 0;
end;
```

**Listagem 7.** Recebendo uma mensagem

```
procedure TfrmCliente.ClientSocketRead
(Sender: TObject;
 Socket: TCustomWinSocket);
var
  Texto : String;
begin
  Texto := Socket.ReceiveText;
   Memo.Lines.Add(GetTag(Texto,'<NICK>',
   '</NICK>') + ' diz : ' + getTag(Texto,'<MENSAGEM>',
   '</MENSAGEM>'));
end;
```

**Listagem 8.** Extraindo texto

```
function TfrmCliente.GetTag(Source, TagINI,
TagFN: String): String;
begin
 Result:= MidStr(Source, (Pos(TagINI,Source)+
 Length(TagINI)),((Pos(TagFN,Source)) -
 (Pos(TagINI,Source)+  Length(TagINI))));
end;
```

## Conclusão

A comunicação por Socket pode ser utilizada em várias aplicações que vão além de um chat. Por exemplo em um sistema de atendimento ao cliente onde atendentes precisavam chamar os clientes através de um Display (muito parecido como um sistema de senha de banco).

Uma grande vantagem em utilizar sockets é o fato que o próprio desenvolvedor pode criar o seu próprio protocolo de comunicação, personalizando o texto que é enviado, tornando assim os Sockets uma ótima alternativa para diversas situações. Esperamos que esse QuickUpdate tenha ajudado no seu entendimento do funcionamento dos Sockets.

**Rafael Isique Guardieri Brendolan**
*rafaelbrendolan@hotmail.com*
Cursando Sistemas de Informação pela São Luis - Jaboticabal. Atua como análista e programador de sistemas da equipe de desenvolvimento Delphi da Siplan Control-M (www.siplancontrolm.com.br).

**Wanderley Donizeti Panosso Junior**
*wanderley_panosso@hotmail.com*
Cursando Análise de Desenvolvimento de Sistemas pela FATEC-TQ. Atua como análista e programador de sistemas da equipe de desenvolvimento Delphi da Siplan Control-M (www.siplancontrolm.com.br).


**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Mini curso NF-e e Delp

## Uma solução passo a passo – Par

### Resumo DevMan

***De que trata o artigo:***

Esta série de artigos demonstrará como utilizar uma aplicação criada na nova plataforma da Embarcadero, a FireMonkey, para gerar, assinar e validar notas fiscais eletrônicas partindo do zero, sem a utilização de componentes de terceiros integrados à ferramenta, bem como consumir os recursos dos Web Services da Secretaria da Fazenda para o envio e consulta das mesmas, finalizando com a impressão da DANFE. Nesta primeira parte são abordados os itens necessários para a geração da NF-e.

***Em que situação o tema é útil:***

Possibilitar o desenvolvimento de um módulo de nota fiscal eletrônica para ser integrado as aplicações desenvolvidas na plataforma nova FireMonkey, uma vez que a mesma não tenha ainda componentes voltados para a utilização da NF-e, bem como demonstrar passo á passo o desenvolvimento de uma aplicação NF-e.

***FireMonkey e NF-e 2.0:***

Atualmente grande parte das empresas de vários seguimentos é obrigada a emitir a nota fiscal eletrônica e muitas podem optar pela mesma. Por este motivo, os novos e antigos softwares comerciais e industriais devem possibilitar uma estrutura para a elaboração e transmissão das mesmas, caso contrário correm sérios riscos de perder mercado. Como a FireMonkey ainda é uma plataforma muito recente, evidentemente não possui tantos componentes para funções especificas em comparação com a plataforma VCL, como é o caso da integração com a NF-e. Nesta primeira parte do artigo é abordada a estrutura básica para a geração do arquivo XML da NF-e através do recurso XML Data Binding, que em suma, permite que um documento ou XML Schema possa ser importado para a aplicação de uma maneira muito rápida e prática, onde tal recurso também esta presente em diversas versões do Delphi, bem como a elaboração das funções básicas para a geração da nota fiscal eletrônica, como é o caso de remoção de caracteres não permitidos, geração de número aleatório, cálculo do digito verificador e elaboração da chave de acesso. Para concluir, um exemplo completo da composição de uma nota fiscal por uma empresa optante pelo Simples Nacional.

A Nota Fiscal Eletrônica se tornou uma obrigatoriedade na maioria das aplicações comerciais e industriais da atualidade no país. Ela surgiu com o propósito de substituir a emissão de notas fiscais em papel por um documento eletrônico, devidamente baseado em um padrão constituído de várias regras e que deve ser assinado digitalmente pelo emitente através de um certificado digital para garantir sua autenticidade. Tal documento deve ser elaborado, validado, transmitido, autorizado e por fim, consultado através da Internet, possibilitando assim um acompanhamento fiscal das movimentações comerciais praticamente em tempo real pela Secretaria da Fazenda - SEFAZ. Entretanto, para o transporte das mercadorias presentes na nota e a facilidade de consulta, existe o DANFE (Documento Auxiliar da Nota Fiscal Eletrônica), que pode ser impresso em uma folha de papel comum, porém que não substitui o documento eletrônico e tampouco é considerado uma nota fiscal.

Há alguns anos atrás quando surgiu, a Nota Fiscal Eletrônica fez com que muitas

# hi XE 2
# rte 1

Giuliano Scombatti Pinto

empresas e desenvolvedores tivessem que se adaptar e adotar um novo padrão na elaboração de seus projetos e alimentação dos dados nas bases.

Além do executável da aplicação comercial, suas informações também devem estar coerentes e compostas pelos objetos que estão presentes na nota fiscal, como os impostos, para que os mesmos possam ser futuramente calculados. Também é importante ressaltar que para a emissão das notas fiscais, a empresa deve entrar em contato com um órgão responsável para a obtenção do certificado digital, possibilitando assim a futura assinatura da nota. É evidente que esta se torna a primeira fase da implantação do Projeto de NF-e, ou seja, a base necessária para que uma nota possa ser gerada e assinada pela aplicação.

Uma vez que a base para o Projeto está concluída, os dados das notas fiscais devem ser sujeitos a uma validação ainda na aplicação do cliente, permitindo assim a detecção de erros antes do envio das mesmas. Tal validação deve ser feita através dos schemas disponibilizados pela Secretaria da Fazenda. Os schemas (arquivos XSD – XML Schema Definition) são documentos baseados em XML (Extensible Markup Language) que possuem regras de validação para serem aplicadas em arquivos XML, uma vez que a nota fiscal eletrônica é um documento baseado nesta linguagem. Algumas regras de validação que devem ser bem observadas durante a elaboração da nota fiscal eletrônica, pois podem causar "dores de cabeça" no momento da validação, tratam da existência de espaços no início e no fim das informações presentes nos elementos, bem como a utilização de quebras de linha no arquivo XML com a intenção de efetuar uma formatação no arquivo texto, dentre várias outras.

Com o documento eletrônico elaborado, assinado e validado, o próximo passo é efetuar sua transmissão para um Web Service da Secretaria da Fazenda. Um Web Service é um serviço automatizado que disponibiliza as mais variadas funções através da Internet, permitindo assim a troca de informações entre diversas aplicações e plataformas, sendo baseado em XML e HTML. Em suma, ele "simplesmente" recebe uma solicitação, processa e devolve uma resposta para o solicitante. Para que exista tal comunicação, uma implementação do protocolo SOAP (Simple Object Access Protocol) pode ser efetuada, disponibilizando assim a independência necessária para o funcionamento do Web Service. Contudo, para que se possa acessar e utilizar os recursos de um Web Service, há a necessidade de se conhecer sua estrutura. Tal estrutura pode ser obtida através de um arquivo XML sob a padronização WSDL (Web Service Description Language) ou, muitas vezes, através de um manual de utilização do mesmo, como é o caso dos Web Services da Secretaria da Fazenda. Para o ambiente da Nota Fiscal Eletrônica são disponibilizados vários Web Services, sendo destinados para homologação (testes) e produção. Estes possibilitam o envio, cancelamento, inutilização e diversas consultas das notas fiscais eletrônicas emitidas através da aplicação cliente. Vale à pena lembrar que os Web Services também são constituídos de serviços síncronos e assíncronos, onde neste último, a aplicação cliente deve fazer futuras consultas até obter a resposta almejada. No que diz respeito ao FireMonkey, a mais recente plataforma de desenvolvimento de aplicações presente no Delphi XE 2, também pode ser utilizada para a elaboração das notas fiscais eletrônicas. Vale a pena lembrar que vários objetos e recursos presentes na VCL também estão presentes no FireMonkey, como é o caso do XML Data Binding, que será abordado posteriormente, possibilitando assim a estrutura para a elaboração de uma aplicação NF-e.

### Nota do DevMan

O XML (Extensible Markup Language) é um documento que utiliza uma linguagem de marcação de acordo com a W3C para ser utilizado na descrição e armazenamento dos mais diversos tipos de informações, visando o compartilhamento de dados principalmente através da Internet, e como seu próprio nome sugere, é extensível, não havendo um limite pré-estabelecido para o número de Tags que podem ser utilizadas, bem como as mesmas podem ser criadas pelo próprio desenvolvedor do arquivo. Possui uma composição baseada em elementos também conhecidos como "nós", sendo constituído de um elemento raiz (nó raiz) e vários elementos filhos de acordo com a necessidade do projeto. Vale à pena evidenciar que tais nós podem possuir atributos, bem como podem utilizar uma Tag inicial '< >' e uma Tag final '< />' para a estrutura dos seus dados. Para concluir, um arquivo XML é basicamente um arquivo de texto evidentemente estruturado, que pode ser simplesmente alterado por qualquer editor de textos.

### Nota do DevMan

A W3C (World Wide Web Consortium) é um consórcio internacional com vários membros, envolvendo diversas instituições cujo objetivo é estabelecer padrões para a elaboração e interpretação de vários documentos utilizados na Web. Em outras palavras, um site ou um arquivo XML que é criado sob estes padrões, pode ser acessado por qualquer Software interpretador, permitindo assim que computadores, Tablets, Smartphones entre outros, acessem e obtenham tal conteúdo normalmente, independente da plataforma utilizada.

## XML Data Binding

O recurso XML Data Binding é muito útil e se torna essencial para a elaboração de uma aplicação NF-e, obtendo assim uma alta

produtividade, uma vez que a Secretaria da Fazenda disponibiliza os XML Schemas para geração e validação das notas fiscais.

O assistente para utilização deste recurso pode ser facilmente acessado no Delphi XE 2 através do menu File>New>Other>XML>XML Data Binding. Vale à pena evidenciar que o XML Data Binding também está disponível em versões anteriores do Delphi. Através dele é possível gerar classes originadas de XML Schemas (XSD), arquivos DTD (Document Type Definition), Data Reduced XML (XDR) e etc.

Ao utilizar o assistente, a primeira etapa é a seleção do esquema ou documento XML para que futuramente uma Unit possa ser gerada automaticamente de acordo com o Schema informado, permitindo também que possam ser especificadas as estratégias de nomes para a geração das interfaces e classes. A **Figura 1** demonstra a tela inicial do assistente de XML Data Binding do Delphi XE 2.

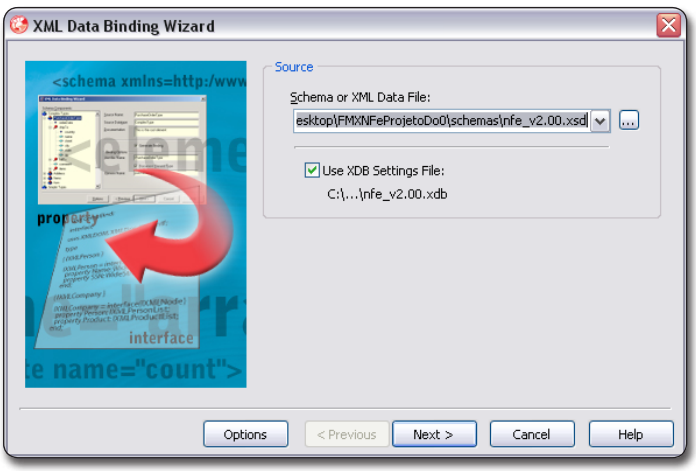



**Figura 1.** Tela inicial do assistente de XML Data Binding

No próximo passo, é possível visualizar e fornecer informações detalhadas para cada nó e seus elementos (se existirem) no esquema ou documento. Também pode ser gerada uma função global que utiliza um objeto do tipo TXMLDocument, possibilitando o retorno da interface do nó raiz pertencente a hierarquia de dados. O TXMLDocument é um objeto que pode ser utilizado para a construção de um arquivo XML (novo ou existente), bem como possibilitar sua leitura, podendo ser futuramente gravado em um arquivo.

Com a conclusão do assistente, o documento XML é abstraído para um grupo de interfaces que permitem a definição de valores finais, em outras palavras, a representação de um objeto (documento XML) é feita na memória, não necessitando que a aplicação utilize DOM (Document Object Model), SAX (Simple API for XML) ou outro recurso para acesso aos dados. Ao utilizar tal recurso, o código do nó raiz é uma interface derivada de TXMLDocument, como mencionado anteriormente, e seus nós filhos podem ser derivados de qualquer IXMLNode ou interface gerada à partir do tipo do nó. Após a obtenção das classes, a leitura e a criação de arquivos do tipo XML se torna muito mais fácil e rápida, permitindo uma maior produtividade e flexibilidade na manipulação dos dados. Vale à pena evidenciar também que, como o XML foi mapeado em uma Unit, a mesma pode ser facilmente gravada em conjunto com o projeto. Como pode ser observado, na elaboração das classes o Delphi utiliza interfaces para a geração dos métodos públicos, podendo assim utilizar o mecanismo denominado contagem de referência para que a instância seja liberada da memória de uma maneira automática, sendo assim, um método que utilize a interface não necessita que sua referência seja removida de forma explicita.

Contudo, o uso do assistente (XML Data Binding) pode não ser indicado quando existe a necessidade do carregamento de todos os documentos XML na memória, onde o ideal é a utilização dos recursos DOM.

**Nota do DevMan**

A utilização de Interfaces é muito útil, pois em suma, permite que códigos sejam escritos independentemente da implementação. As interfaces possuem um identificador denominado GUID que é uma constante composta de 128 bits, bem como seus próprios métodos. Uma vez que as classes podem implementar diversas interfaces, quando a aplicação está em execução, o GUID é utilizado para a obtenção de um ponteiro para a implementação da interface para que se possa descobrir se um objeto utiliza a mesma.

**Nota do DevMan**

A contagem de referência é um mecanismo que permite que seja armazenado o número de ponteiros ou referências para determinado recurso, como um objeto. Sendo assim, se determinado objeto não é mais referenciado, ele pode ser destruído. O Delphi também implementa tal mecanismo para poder criar e destruir determinados objetos de uma maneira automatizada.

## Tipos de Emissão da NF-e

Como seu próprio nome sugere, a NF-e é um documento eletrônico que é transmitido através da Internet, como mencionado anteriormente. A validade da Nota e seu respectivo DANFE dependem da autorização de uso dos mesmos. Com certeza a alta disponibilidade é uma das prioridades dos sistemas de recepção da NF-e. Evidentemente, para ser elaborada e autorizada, a nota vai depender de diversos recursos de Hardware e Software (o envio através da Internet, o status dos Web services e etc) e podem existir instabilidades ou mau funcionamento dos mesmos, ocasionando assim uma indisponibilidade da emissão da nota como é feita habitualmente, podendo prejudicar ou atrasar as entidades envolvidas (emitente, transportadora e destinatário). Sendo assim, existe a necessidade da implementação de métodos alternativos. A **Tabela 1** descreve os tipos de emissão da NF-e.

Quando a NF-e é emitida, a sua modalidade de emissão deve ser informada, levando em consideração qual é a forma mais conveniente. As modalidades são representadas por números na NF-e, podendo ser 1 para normal, 2 para contingência FS, 3 para contingência SCAN, 4 para DPEC e 5 para contingência FS-DA.

| Tipo | Descrição |
|---|---|
| Normal | É o utilizado habitualmente para a autorização de uso da NF-e, permitindo assim a impressão do DANFE, evidentemente quando toda a estrutura envolvida esta disponível. |
| FS | A contingência com uso do formulário de segurança é uma alternativa mais comum para situações onde não é possível obter a autorização do uso da NF-e por indisponibilidade de algum dos recursos envolvidos, como é o caso de problemas com a conexão de Internet. Neste caso, a NF-e pode ser emitida em contingência e a impressão do DANFE pode ser realizada no formulário de segurança. Sendo assim, quando a situação é normalizada, as notas fiscais devem ser enviadas. Contudo, apenas empresas que possuam o FS podem utilizar este meio. |
| FS-DA | A contingência com uso do formulário de segurança para impressão de documento auxiliar do documento fiscal eletrônico é mais uma alternativa que pode ser adotada durante instabilidades. Seu funcionamento é equivalente ao uso do formulário de segurança, porém, ao invés do uso de FS utiliza-se o FS-DA. |
| SCAN | O sistema de contingência do ambiente nacional também é muito utilizado, pois, permite a elaboração da NF-e e a impressão do DANFE em papel comum, não havendo a necessidade da transmissão da NF-e para a Secretaria quando o problema for estabilizado. Vale à pena evidenciar que esta modalidade conta com uma numeração de série específica (900 – 999) para sua emissão. Entretanto o SCAN só é ativado quando existe algum problema com a recepção normal da NF-e pelo SEFAZ. |
| DPEC | A declaração prévia de emissão em contingência é mais uma alternativa em meio a instabilidades. Neste caso, é feito um prévio registro resumido das notas emitidas. É permitida a impressão do DANFE, porém a NF-e deverá ser transmitida assim que possível. |

**Tabela 1.** Tipos de emissão da nota fiscal eletrônica

| Nº de dígitos | Descrição |
|---|---|
| 02 | Deve conter o código da UF (IBGE) do emitente. Exemplo: 35 (código do estado de SP). |
| 04 | Deve conter o ano abreviado e o mês da emissão (formato YYMM). |
| 14 | Deve conter apenas os números do CNPJ do emitente, sem qualquer caractere de formatação. |
| 02 | Modelo do documento (55). |
| 03 | Série do documento (normalmente 001 ou 900 à 999 quando em contingência SCAN). |
| 09 | Número da NF-e, campo de incremento gerado pela aplicação cliente. |
| 01 | Modalidade ou tipo de emissão (1 = normal). Obs: Não deve ser confundido com tipo de ambiente (homologação e produção). |
| 08 | Código numérico gerado aleatoriamente pela aplicação. |
| 01 | Digito verificador da chave calculado com base no Módulo 11. |

**Tabela 2.** Composição da Chave de Acesso da NF-e

Entretanto, o emitente pode contar com a utilização de apenas uma modalidade, várias ou todas. É importante evidenciar que o manual de contingência também é um material importante que deve ser lido para poder elaborar um projeto de emissão em contingência de NF-e.

## Início do projeto

Para dar início ao projeto de NF-e, levando em consideração que já exista uma estrutura básica de sistema (um projeto em FireMonkey com banco de dados que possua cadastro de clientes, produtos, CFOP, situação tributária, municípios e etc), os XML Schemas da NF-e versão 2.0 mais atualizados devem ser adquiridos do site da Secretaria da Fazenda. Também é essencial a obtenção do manual de integração do contribuinte para auxiliar em vários momentos na elaboração da aplicação NF-e. Posteriormente, é criada uma nova aplicação FireMonkey. Antes da elaboração da nota fiscal, são criadas algumas funções para serem utilizadas no desenvolvimento. Um dos primeiros passos necessários para a elaboração da nota é a geração da chave de acesso. A chave de acesso deve ser criada pela aplicação cliente de acordo com as especificações da Secretaria da Fazenda. Além de presente no documento XML, ela deve ser usada para o nome da NF-e, sendo constituída na forma: "[chave de acesso]-nfe.xml". Vale a pena evidenciar que existem padrões para elaboração dos nomes dos arquivos, descritos no manual do contribuinte, visando uma maior facilidade para os "diversos estados" que a NF-e (ou arquivo XML) se encontra. A **Tabela 2** descreve a composição da chave de acesso da NF-e que deve ser gerada pela aplicação.

Como pode ser observado na **Tabela 1**, o número total de dígitos da chave de acesso é equivalente a 44 (43 da chave gerada + 1 dígito verificador). Vale à pena lembrar que em casos onde os números não ocupam o tamanho total estabelecido, como o número da NF-e (09 dígitos) que é um código incremental, a aplicação deverá completar tal tamanho utilizando zeros à esquerda. A **Listagem 1** demonstra uma simples função que pode ser utilizada para inserir zeros à esquerda de determinado número, levando em consideração a quantidade atual de dígitos do mesmo, bem como a quantidade de dígitos que se deseja obter.

Concluída a função que insere os zeros à esquerda, deve-se criar uma função que limpe o formato de uma determinada String devolvendo apenas os números para que possa ser aplicada no CNPJ e posteriormente em outros campos, como é o caso do código do IBGE do município e do CEP, onde o caractere "-" deve ser removido. Evidentemente existem várias formas de se remover ou substituir caracteres no Delphi. Tal recurso pode ser obtido através do uso da função StringReplace() por exemplo.

Esta função é muito útil, porém só permite a substituição ou remoção de um determinado caractere por vez, devendo ser repetida quando necessitar remover mais caracteres. Contudo, a **Listagem 2** ilustra uma função para a remoção de vários caracteres indesejados simultaneamente e sem a necessidade de utilizar StringReplace().

Com as funções básicas criadas, o próximo passo é gerar o número aleatório da nota fiscal. Tal número presente na elaboração da chave de acesso deve ser gerado pela aplicação para cada NF-e com o objetivo de evitar acessos indevidos. A **Listagem 3** demonstra um exemplo de função para gerar os 8 dígitos necessários aleatoriamente.

**Listagem 1.** Função para inserir zeros à esquerda de um número

```
function InserirZerosEsquerda(Numero, QtdeZeros: Integer): String;
var
  NZeros: String;
  i: Integer;
begin
  NZeros := IntToStr(Numero);
  for i := QtdeZeros - Length(NZeros) downto 1 do
    NZeros := '0' + NZeros;
  Result := NZeros;
end;
```

**Listagem 2.** Função para limpar caracteres de uma string

```
function LimparCaracteres(texto: string): string;
var
  i: Integer;
  TextoSemCaracteres: string;
begin
  for i := Length(texto) downto 1 do
    if not (texto[i] in ['','-','/']) then
      TextoSemCaracteres := texto[i] + TextoSemCaracteres;
  Result := TextoSemCaracteres;
end;
```

**Listagem 3.** Função para gerar 8 números aleatórios

```
function gerarNumeroAleatorio: string;
var
  n: string;
  i: Integer;
begin
  n := '';
  for i := 1 to 8 do
  begin
    Randomize;
    n := n + IntToStr(1 + Random(8));
  end;
  Result := n;
end;
```

O próximo passo é criar a função para calcular o digito verificador (Módulo 11). Para tanto, deve-se multiplicar da direita para esquerda, os números da chave de acesso da NF-e que foi gerada, por seus devidos pesos. Os pesos são definidos pela sequência 2, 3, 4, 5, 6, 7, 8, 9. Tal sequência deve ser repetida até que todos os números tenham sido multiplicados. Após a multiplicação, deve-se somar o resultado de todas as ponderações e dividir por 11, obtendo o resto da divisão (somatória mod 11). Para finalizar, o valor do digito verificador é obtido subtraindo 11 do resto da divisão (11 - resto), entretanto se o resto da divisão é 0 ou 1, o digito verificador deverá ser 0. A **Listagem 4** demonstra um exemplo para o cálculo do digito verificador.

Para concluir as funções básicas para a elaboração da NF-e, o próximo item gera a chave utilizando as funções já descritas. Tal função é baseada na **Tabela 2** descrita anteriormente. Neste momento também é interessante implementar um mecanismo para informar se a NF-e é emitida na modalidade 1 (normal) ou se em contingência (2, 3, 4 ou 5). Vale à pena lembrar que caso a NF-e seja emitida em contingência, a modalidade de emissão deve ser informada. A **Listagem 5** demonstra um exemplo da geração da chave de acesso utilizando as funções anteriores. Vale à pena destacar que tais informações são geradas diretamente neste exemplo, contudo o ideal é a utilização de uma Query que busque os dados da tabela de acordo com a NF-e.

**Listagem 4.** Função para cálculo do digito verificador (Módulo 11)

```
function CalcularDigitoMod11(chave: String): String;
const
  pesos: array[1..8] of Integer = (2,3,4,5,6,7,8,9);
var
  soma, resto, i, j, digitoverificador: Integer;
begin
  soma := 0;
  j := 1;
  chave := StringReplace(chave,'',[rfReplaceAll]);
  for i := Length(chave) downto 1 do
  begin
    soma := soma + StrToInt(chave[i]) * pesos[j];
    Inc(j);
    if j > 8 then j := 1;
  end;
  resto := soma mod 11;
  if resto > 1 then
    digitoverificador := 11 - resto
  else
    digitoverificador := 0;
  Result := IntToStr(digitoverificador);
end;
```

**Listagem 5.** Exemplo de função para gerar a chave de acesso da NF-e

```
function gerarChaveAcessoNFe(numeroNFE: Integer; nAleatorio: string): string;
var
  chave: string;
begin
  chave := '35'; //Código IBGE do Estado
  chave := chave + FormatDateTime('YYMM', Date);
  chave := chave + LimparCaracteres('00.000.000/0001-00');
  chave := chave + '55';
  chave := chave + InserirZerosEsquerda(1,3);
  //Levar em consideração a modalidade de emissão
  chave := chave + InserirZerosEsquerda(numeroNFE,9);
  chave := chave + '1'; //Verificar modalidade de emissão
  chave := chave + nAleatorio;
  chave := chave + CalcularDigitoMod11(chave);
  Result := chave;
end;
```

Conforme a **Listagem 5**, o número da NF-e e o número aleatório são passados como parâmetros, pois, os mesmos também deverão estar presentes no conteúdo do XML da NF-e. Uma outra

alternativa seria criar uma variável global para armazenar tais valores e utilizá-los posteriormente.

## Nota Fiscal

Para iniciar com a elaboração do conteúdo da NF-e é preciso efetuar um XML Data Binding no arquivo denominado "nfe_v2.00.xsd", base para a geração das notas fiscais, presente nos Schemas da NF-e. Para tanto, como foi mencionado anteriormente, basta acessar o Data Binding, definir o nome do arquivo citado anteriormente e clicar em Next e continue avançando. Deve-se a opção Store in file para que o assistente crie automaticamente a unit que denominada "nfe_v200.pas". Para dar continuidade, também deve ser criado um formulário para a elaboração das notas fiscais de saída contendo as informações do destinatário da nota, itens, transporte e etc.

A **Listagem 6** demonstra a declaração das variáveis mais comuns em um procedimento para gerar a NF-e, sendo algumas obtidas através da unit "nfe_v200.pas", devendo a mesma ser inclusa na seção uses da unit atual. Os itens que compõem o preenchimento dos elementos desta nota de exemplo foram elaborados com base em uma empresa optante pelo Simples Nacional.

**Listagem 6.** Início da função para gerar o arquivo XML da NF-e

```
procedure gerarNFe200(numNF: Integer);
var
  NFe: IXMLTNFe;
  InfNFe: IXMLInfNFe;
  Ide: IXMLIde;
  Emit: IXMLEmit;
  Dest: IXMLDest;
  Det: IXMLDet;
  Total: IXMLTotal;
  Transp: IXMLTransp;
  InfAdic: IXMLInfAdic;
  NumNFe: Integer;
  Serie: Integer;
  ChaveAcessoNFe: string;
  ProItem: Integer;
  XMLDoc: TXMLDocument;
  NumAleatorioNFe: string;
  ...
```

A próxima etapa na geração da NF-e pode ser a alteração do DecimalSeparator para ".", uma vez que é o padrão que deve ser adotado nos campos de valores decimais da nota fiscal. O DecimalSeparator é utilizado para a definição do caractere que será usado na exibição de variáveis do tipo ponto flutuante. Contudo, se preferir o desenvolvedor pode optar por usar a função FormatFloat(), que também pode ser usada para formatar as variáveis necessárias. Também deve ser criado um objeto do tipo TXMLDocument, possibilitando assim a utilização da função GetDocBinding(). A função GetDocBinding() do objeto TXMLDocument é responsável por substituir o elemento do documento XML por uma instância de uma classe, que deve ser informada como parâmetro. Depois de substituído o nó informado pela instância da classe, a função retorna uma interface relativa ao elemento. A função é constituída por XMLDocument.GetDocBinding ([nome da Tag], [classe do nó], [URI do Namespace]). A **Listagem 7** demonstra a inicialização das variáveis e a atribuição de seus valores de acordo com as especificações da NF-e.

### Nota do DevMan

A Namespace URI é utilizada para evitar conflitos em nomes de elementos em arquivos XML. É um padrão definido pela W3C e deve ser definido no atributo xmlns no início da tag de um elemento. Sendo assim, todos os elementos filhos com o mesmo prefixo são associados ao determinado namespace. No caso da nota fiscal eletrônica, a namespace é http://www.portalfiscal.inf.br/nfe.

### Nota do DevMan

O URI (Uniform Resource Identifier) pode ser definido como uma cadeia de caracteres com o propósito de identificar um recurso presente na Internet. O URI mais encontrado é o URL (Uniform Resource Locator), que é utilizado para a identificação de um domínio na Internet.

**Listagem 7.** Inicialização das variáveis, atribuição e utilização das funções

```
XMLDoc := TXMLDocument.Create(nil);
XMLDoc.Active := True;
NFe := XMLDoc.GetDocBinding('NFe', TXMLTNFe, 'http://www.portalfiscal.inf.br/nfe')
as IXMLTNFe;
InfNFe := NFe.InfNFe;
InfNFe.Versao := '2.00';
NumAleatorioNFE := gerarNumeroAleatorio;
ChaveAcessoNFe := gerarChaveAcessoNFe(numNF, NumAleatorioNFe);
InfNFe.Id := 'NFe'+ ChaveAcessoNFe;
```

Como mostrado na **Listagem 7**, o objeto TXMLDocument (neste caso XMLDoc) é inicializado e ativado. Também é realizado o Binding do documento com as informações de acordo com os Schemas da NF-e. Após a criação do objeto XMLDocument, bem como seu Binding, a próxima etapa é preencher os elementos filhos do elemento "Ide", destinado ao grupo de informações de identificação da NF-e. Neste momento, o resultado das funções utilizadas (como a geração do número aleatório e o cálculo do digito verificador) e outros dados que foram utilizados na a elaboração da chave de acesso da nota também devem estar presentes, assim como outras informações obrigatórias para a NF-e. No manual do contribuinte é possível ter informações detalhadas sobre cada item necessário para a elaboração da NF-e. A **Listagem 8** demonstra o exemplo de preenchimento do elemento "Ide" da nota fiscal eletrônica.

Com as informações de identificação devidamente preenchidas, é preciso preencher dos dados do emitente. Para tanto, basta utilizar o elemento denominado "Emit" e preencher seus elementos filhos com as informações da empresa. A **Listagem 9** demonstra um exemplo de preenchimento do elemento "Emit" da nota fiscal eletrônica. Neste exemplo a função LimparCaracteres(), criada anteriormente, também é utilizada para formatar os números do CNPJ.

**Listagem 8.** Exemplo de preenchimento do elemento "Ide" da NFe

```
Ide := InfNFe.Ide;
with Ide do
begin
  CUF := '35'; //Código UF IBGE;
  CNF := NumAleatorioNFE; //Número aleatório
  NatOp := Trim('REVENDA DE MERCADORIA');
  IndPag := '1'; //Condição de pagamento
  Mod_ := '55';
  Serie := '1'; //Serie da NFe
  NNF := IntToStr(numNF); //Número da nota
  DEmi := FormatDateTime('yyyy-mm-dd',Date); //Data de emissão
  TpNF := '1'; //Tipo do Movimento (entrada = 0, saida = 1)
  CMunFG := '3531308'; //municipio do emitente sem o "-"
  TpImp := '1';
  TpEmis := '1'; //Tipo Emissão NFe
  CDV := CalcularDigitoMod11(Copy(ChaveAcessoNFe,1,Length
  (ChaveAcessoNFe) -1));
  //Dígito calculado do modulo 11
  TpAmb := '2';//homologacao = 2, produção = 1;
  FinNFe := '1';//nota complementar = 2 senao = 1
  ProcEmi := '0';
  VerProc := 'FMX-NFe2 v1';
end;
```

**Listagem 9.** Preenchimento do elemento "Emit" da NFe

```
Emit := InfNFe.Emit;
with Emit do
begin
  CNPJ := LimparCaracteres('00.000.000/0001-00');
  XNome := Trim('EMPRESA X LTDA'); //Razão social
  XFant := Trim('EMPRESA X'); //Nome fantasia
  with EnderEmit do
  begin
    XLgr := Trim('RUA X'); //Logradouro sem o número
    Nro := Trim('000'); //Número
    XBairro := Trim('CENTRO'); //Bairro
    CMun := '3531308'; //Cod municipio IBGE sem o "-"
    XMun := Trim('MONTE ALTO'); //Município
    UF := Trim('SP'); //Estado
    CEP := Trim('15910000'); //CEP sem o "-"
    CPais := '1058';
    XPais := 'Brasil';
    Fone := Trim('1600000000');
  end;
  IE := Trim('000000000000');
  CRT := '1';
end;
```

Na sequência são informados os dados do destinatário da nota. Para tanto, basta utilizar o elemento denominado "Dest" e preencher seus elementos filhos com os valores desejados. A **Listagem 10** demonstra o preenchimento do destinatário.

Concluído o preenchimento dos dados de identificação, do emitente e do destinatário, chegou a vez de preencher as informações de cada produto. O elemento da NF-e destinado aos produtos é denominado "Det" (detalhamento de produtos e serviços da NF-e). Neste momento devem ser preenchidos os valores dos impostos como o ICMS, IPI, COFINS, entre outros. A **Listagem 11** ilustra o preenchimento de um determinado produto como exemplo, lembrando que os impostos são baseados em uma empresa optante do Simples Nacional, portanto, estes valores podem sofrer modificações de acordo com a modalidade fiscal.

**Listagem 10.** Preenchimento do elemento "Dest" da NFe

```
Dest := InfNFe.Dest;
with Dest do
begin
  CNPJ := Trim('00000000000100');
  XNome := Trim('NF-E EMITIDA EM AMBIENTE DE HOMOLOGACAO -
 SEM VALOR FISCAL');
  with EnderDest do
  begin
    XLgr := Trim('ENDERECO X');
    Nro := Trim('SN');
    XBairro := Trim('BAIRRO X');
    CMun := '3531308';
    XMun := 'MONTE ALTO';
    UF := 'SP';
    CEP := Trim('15910000');
    CPais := '1058';
    XPais := 'Brasil';
    Fone := Trim('1600000000');
  end;
  IE := Trim('000000000000');
end;
```

**Listagem 11.** Preenchimento do elemento "Det" da NFe

```
ProItem := 1;
Det := InfNFe.Det.Add;
with Det do
begin
  NItem := IntToStr(ProItem);//Número do item
  with Prod do
  begin
    CProd := Trim('0001.0001');//Código do produto
    CEAN := '';
    XProd := Trim('PRODUTO X');
    NCM := Trim('00000000');
    CFOP := Trim('5405');
    UCom := 'PT'; //Unidade de medida
    QCom := '10.00';
    VUnCom := '5.00';
    VProd := '50.00';
    CEANTrib := '';
    UTrib := 'PT';
    QTrib := '10.00';
    VUnTrib := '5.00';
    IndTot := '1';
  end;
  with Imposto do
  begin
    with ICMS.ICMSSN500 do
    begin
      Orig := '0';
      CSOSN := '500';
      VBCSTRet := '0.00';
      VICMSSTRet := '0.00';
    end;
    with IPI do
    begin
      CEnq := '999';
      IPINT.CST := '53';
    end;
    with PIS.PISOutr do
    begin
      CST := '99';
      VBC := '50.00';
      PPIS := '0.00';
      VPIS := '0.00';
    end;
    with COFINS.COFINSOutr do
    begin
      CST := '99';
      VBC := '50.00';
      PCOFINS := '0.00';
      VCOFINS := '0.00';
    end;
  end;
end;
```

Inseridos todos os itens, a próxima etapa é informar os valores totais da nota. O elemento responsável por tal tarefa também é denominado "Total". A **Listagem 12** exemplifica o preenchimento de tal elemento.

Para concluir o exemplo de nota, bastam apenas os dados da transportadora. Para tanto é utilizado o elemento denominado "Transp". A **Listagem 13** exemplifica seu preenchimento.

**Listagem 12.** Preenchimento do elemento "Total" da NFe

```
Total := InfNFe.Total;
with Total.ICMSTot do
begin
  VBC := '0.00';
  VICMS := '0.00';
  VBCST := '0.00';
  VST := '0.00';
  VProd := '50.00';
  VFrete := '0.00';
  VSeg := '0.00';
  VDesc := '0.00';
  VII := '0.00';
  VIPI := '0.00';
  VPIS := '0.00';
  VCOFINS := '0.00';
  VOutro := '0.00';
  VNF := '50.00';
end;
```

**Listagem 13.** Preenchimento do elemento "Transp" da NF-e

```
Transp := InfNFe.Transp;
with Transp do
begin
  ModFrete := '0';
  with Transporta do
  begin
    CNPJ := LimparCaracter('00.000.000/0001-00');
    XNome := Trim('TRANSPORTADORA X');
    IE := Trim('000000000000');
    XEnder := Trim('ENDERECO X SN');
    XMun := Trim('MONTE ALTO');
    UF := 'SP';
  end;
end;
```

Depois de preenchidos todos os elementos necessários para este tipo de nota, para finalizar é necessária a gravação do arquivo XML construído para um diretório em disco para que futuramente possa ser assinado, validado, enviado e etc. Neste caso, o interessante é criar um diretório dentro da pasta do sistema ou onde for desejado e atribuir o nome de "Enviar", para receber todas as notas fiscais que ainda precisam ser assinadas e enviadas. A código a seguir demonstra a etapa final na elaboração da NF-e, gravando o arquivo XML no disco rígido.

```
XMLDoc.SaveToFile('C:\FMXNFE\NfeEnviar\' + ChaveAcessoNFe + '-nfe.xml');
XMLDoc.Active := False;
DecimalSeparator :=',';
```

## Conclusão

A Nota Fiscal Eletrônica é uma necessidade para as aplicações da atualidade. Ela pode ser facilmente integrada com uma aplicação baseada em FireMonkey, utilizando os diversos recursos disponibilizados pelo próprio Delphi para aumentar a produtividade, não havendo a necessidade da utilização de componentes integrados a ferramenta.

Esta primeira parte do artigo aborda a criação de uma NF-e que é um processo complexo (uma vez que envolve cálculo de impostos, obrigatoriedade do preenchimento de certos elementos de acordo com a nota e etc) e que necessita de diversas informações como base. De fato, como a plataforma FireMonkey apresenta um futuro promissor e vários recursos atrativos, como é o caso das animações e recursos 3D que podem ser adicionados à aplicação, vale à pena investir na elaboração de diversos projetos com a mesma.

No próximo artigo serão abordados temas como assinatura da NF-e, validação e consumo de vários Web Services da Secretaria da Fazenda.

**Giuliano Scombatti Pinto**
**giuliano@sygnux.com.br**
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.


**Download do Manual do Contribuinte e dos Schemas**
*https://www.fazenda.sp.gov.br/nfe/downloads/arquivos_vigentes.asp*

**Download do Manual de Contingência**
*http://www.fazenda.gov.br/confaz/confaz/Diversos/ATO_COTEPE_14-09_-_MANUAL_DA_CONTINGENCIA.pdf*

**DevSpace do Autor**
*http://www.devmedia.com.br/space.asp?id=283274*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Acesso a WebServices
## Consumindo webservices atrav

## Resumo DevMan

***De que trata o artigo:***

Este artigo aborda o consumo de WebServices através de aplicações desenvolvidas no Delphi XE 2, utilizando a plataforma FireMonkey ou a VCL. Em um primeiro momento é apresentado um exemplo de como consumir WebServices utilizando o componente HTTPReqResp e posteriormente o mesmo serviço utilizando HTTPRIO. Para concluir é demonstrado como acessar páginas da Web e preencher formulários através da aplicação.

***Em que situação o tema é útil:***

Possibilitar a integração de uma aplicação de usuário com serviços disponibilizados na Internet para consultar e obter as informações desejadas, utilizando mensagens padrões SOAP, a base para o uso dos WebServices.

***WebServices:***

A Internet é um ambiente que oferece as mais variadas informações e possibilidades de comunicação para as pessoas e as organizações. Sendo assim, tal tecnologia pode ser empregada para prover diversos serviços automatizados às aplicações de usuário, utilizando padrões pré-definidos, possibilitando assim a implantação dos WebServices. Atualmente eles são muito comuns, uma vez que permitemagilizar e automatizar as mais variadas tarefas. O Delphi conta com diversos componentes que podem ser utilizados tanto na elaboração quanto no acesso a WebServices, bem como objetos que podem ser utilizados para a troca de informações entre aplicações que se encontram em uma rede interna. Este artigo apresenta o exemplo do consumo de um WebService para obter a temperatura de determinado local de duas maneiras, utilizando HTTPReqResp e HTTPRIO, abordando também o uso do documento WSDL.

Atualmente a Internet é um dos meios de comunicação mais utilizados no mundo, uma vez que estápresente no cotidiano de grande parte das pessoas.Ela torna possível e viável a troca das mais diversas informações entre entidades que estão distantes umas das outras com uma velocidade considerável. O WebService é um recurso que utiliza esta mesma base para prover os mais diversos serviços através da Internet, gratuitos ou pagos. Ele é umsistemaautossuficiente que recebe a solicitação de uma aplicação cliente, processa e retorna uma resposta com relação a tal requisição de uma maneira automática. Por ser baseado em padrões comuns, mais especificamente nas tecnologias HTTP (HyperTextTransferProtocol) e XML (ExtensibleMarkupLanguage), permite a integração e troca de dados entre os mais variados ambientes e plataformas, porém vale à pena evidenciar que um WebService não é direcionado a oferecer uma interface gráfica para visualização das informações por parte dos usuários, mas sim atender as requisições efetuadas pelas aplicações clientes.

O WebService utiliza uma implementação do protocolo SOAP (SimpleObjectAccess Protocol) que é um padrão da W3C, provendo serviços que podem ser utilizados pelos desenvolvedores na elaboração das aplicações, permitindo assim a chamada de métodos através da

# és do Delphi XE 2

Giuliano Scombatti Pinto

linguagem XML. O SOAP é um protocolo que foi projetado para a chamada de aplicações remotas com base em RPC (Remote Procedure Calls), assim como a troca de dados entre as entidades envolvidas. Ele possui diversas versões e é baseado em XML, onde deve ser utilizado em conjunto com o protocolo HTTP para possibilitar o transporte dos dados. Contudo, existe um isolamento entre o banco de dados e os serviços, uma vez que os dados são encapsulados pelo protocolo SOAP.

Para conhecer os serviços ou métodos disponibilizados pelo Web Service, é utilizado um documento XML padronizado de acordo com o WSDL (Web Service DescriptionLanguage). Ele disponibiliza detalhes do funcionamento dos serviços e normalmente é público. Para ser acessado e identificado, assim como outros recursos da Internet, o WebService possui umaURL (UniformResourceLocator), que é simplesmente o endereço do mesmo.

A implementação de um WebService pode ser efetuada sobre as mais diversas tecnologias, inclusive através do Delphi. O WebService pode ser baseado em ASP (Active Server Pages) e classes .NET, Tomcat e framework Axis, PHP e módulo SOAP Extension, dentre outros ambientes. O Tomcat é muito conhecido e utilizado pelos desenvolvedores Java para a criação de servidores para páginas de Internet no formato JSP (Java Server Pages) e consequentemente, WebServices. Já o Axis é um grupo de ferramentas que provêdiversas funcionalidades para a elaboração de um WebService, como a implementação do protocolo SOAP. Evidentemente, um WebService também pode estar interligado a uma base de dados interna para prover as consultas solicitadas.

Quanto à questão de segurança, o WebService pode ser desenvolvido com a utilização de SSL (Secure Socket Layer), o que possibilita a autenticação de usuários, a privacidade dos serviços oferecidos e a integridade das informações. Normalmente o mecanismo de segurança adotado para WebServices é o SSL sobre HTTP, denominado HTTPS (HyperTextTransferProtocolSecure). Contudo, tal mecanismo é um pouco mais lento que as implementaçõesnormais em HTTP. Também podem ser utilizadas assinaturas digitais para garantir a integridade de arquivos baseados em XML (XML Signature). Tal padrão também foi definido pela W3C e possui diversas vantagens, pois também são independentes da plataforma.

Dentre as diversas vantagens na utilização dos WebServices estão: a possibilidade da validação de cartões de crédito, o cálculo de frete baseado em informações como o CEP (muito utilizado pelas lojas virtuais), envio e consulta de notas fiscais eletrônicas (Secretaria da Fazenda), etc. Uma outra vantagem evidente na utilização dos WebServices é a velocidade na obtenção de determinada informação que outrora poderia levar mais tempo para ser adquirida.

Com tudo, os WebServices também apresentam algumas desvantagens. Por serem serviços eletrônicos, podem aparecer problemas de hardware e software, bem como problemas com a rede ou Internet. Sendo assim, a utilização de WebServices não é indicada para aplicações críticas que necessitam que o serviço esteja sempre operando.

## Acessando WebServices

O Delphi possui diversos objetos e recursos que podem ser utilizados para possibilitar a comunicação entre aplicações e osWebServices. Além de permitir tal comunicação, o Delphi também permite a implementação de um WebService através do menu File>New>Other>SOAP Server Application. Como foi mencionado anteriormente, o WebService disponibiliza diversos serviços através da Internet e é baseado no protocolo SOAP, que por sua vez, é baseado em XML e utiliza HTTP, sendo assim, uma mensagem baseada no protocolo SOAP é constituída basicamente de texto organizado e estruturado, sendo dividida em três elementos. A **Tabela 1** descreve os elementos presentes em uma mensagem SOAP.

Entretanto para a utilização de SOAP outras informações também devem estar presentes. São elas a URI alvo, o método que será executado, os parâmetros de requisição ou resposta, a assinatura

| Elemento | Descrição |
|---|---|
| Envelope | Deve estar presente em toda mensagem SOAP, sendo o elemento raiz do XML baseado no mesmo. Também pode possuir atributos adicionais e declaração de NameSpaces URI, bem como o estilo de codificação. |
| Header | Este elemento é o cabeçalho da mensagem, entretanto é opcional. Pode conter dados adicionais, como a partir de qual nó a mensagem deverá ser processada. Quando é utilizado na mensagem, o Header deve estar presente como o primeiro elemento no Envelope. |
| Body | É um elemento obrigatório neste padrão que conterá todos os dados, em outras palavras, a informação que deve ser transportada. Também pode conter um elemento denominado "Fault" para o carregamento de mensagens de erros e status retornadas no momento do processamento. |

**Tabela 1.** Elementos básicos presentes em uma mensagem baseada em SOAP

que pode ser opcional e também como comentado anteriormente, um cabeçalho opcional. A **Listagem 1** demonstra um exemplo de uma mensagem baseada em SOAP versão 1.2.

### Nota do DevMan

A W3C (World Wide Web Consortium) é um consórcio internacional com vários membros envolvendo diversas instituições cujo objetivo é estabelecer padrões para a elaboração e interpretação de vários documentos utilizados na Web. Em outras palavras, um site ou um arquivo XML que é criado sob estes padrões, pode ser acessado por qualquer software interpretador, permitindo assim que computadores, tablets, smartphones, etc, acessem e obtenham tal conteúdo normalmente, independente da plataforma utilizada.

### Nota do DevMan

O Namespace URI é utilizado para evitar conflitos em nomes de elementos em arquivos XML. É um padrão definido pela W3C e deve ser definido no atributo xmlns no início da Tag de um elemento. Sendo assim, todos os elementos filhos com o mesmo prefixo são associados ao determinado namespace. No caso da nota fiscal eletrônica, o namespace é http://www.portalfiscal.inf.br/nfe.

**Listagem 1.** Exemplo de mensagem SOAP

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
    <soap12:Header>
    <!-- Informações do cabeçalho -->
    </soap12:Header>
    <soap12:Body>
        <!-- Método que será executado,Dados -->
    </soap12:Body>
</soap12:Envelope>
```

Para utilizar um WebService existe a necessidade de se conhecer seu funcionamento, bem como os métodos disponibilizados pelo mesmo. O documento WSDL (Web Service DescriptionLanguage) é a linguagem utilizada para descrever o funcionamento do WebService. Ele possui informações que permitem identificar o formato dos métodos que serão invocados, bem como seus parâmetros. O WSDL também é uma linguagem baseada em XML. Através dele, o WebService define todas suas operações, esquemas, interfaces, etc.

O UDDI (Universal Description, Discovery and Integration) também é adotado como uma especificação utilizada para definir um serviço, contendo dados sobre implementações, serviços e provedores.

Com relação aos objetos do Delphi que podem ser utilizados para acessar um WebService ou recursos de rede, a **Tabela 2** demonstra um resumo sobre os mesmos.

## HTTPReqResp

O Objeto HTTPReqResp é um componente que pode ser utilizado para executar uma chamada de um método que pertence a uma interface através do envio de uma mensagem SOAP para um WebService. Quando utilizado no ambiente Windows, o HTTPReqResp faz uso da biblioteca wininet.dll que é encontrada por padrão na pasta System32, sendo assim, o ambiente da aplicação deve conter esta biblioteca para que o componente possa funcionar.

Para a utilização do HTTPReqResp é de total importância a configuração das propriedades do mesmo de acordo com a necessidade do WebService, utilizando-se do documento WSDL ou simplesmente de um manual de utilização do mesmo.

Para dar início ao consumo de WebServices através do Delphi utilizando o componente HTTPReqResp, foi criada uma nova aplicação VCL FormsApplication. O WebService que será consumido neste exemplo é um serviço gratuito que pode ser utilizado para consultar o clima de diversas cidades do mundo. É um recurso disponibilizado pelo site http://www.webservicex.net. O endereço do WebService é http://www.webservicex.net/globalweather.asmx. Foi inserido dois Labels e dois Edits no formulário para permitir que sejam digitados o País e a cidade na qual se deseja efetuar a busca. Também foi inserido um botão para efetuar a consulta ao WebService. A **Listagem 2** demonstra um exemplo de código que foi inserido no evento OnClick do botão para efetuar a pesquisa no WebService através de HTTPReqResp e obter somente o valor da temperatura de determinada cidade informada.

Como pode ser observado no código, é criada uma mensagem no padrão SOAP diretamente em texto (uma vez que se conheça o funcionamento do WebService), tendo seu conteúdo armazenado na variável denominada "msoap". Posteriormente é criado o objeto THTTPReqResp (ReqResp) e são definidas as propriedades

| Objeto | Descrição |
|---|---|
| HTTPRIO | É utilizado para a obtenção de uma referência de determinada interface registrada, permitindo a implementação da mesma. |
| HTTPReqResp | É utilizado para efetuar chamadas de métodos da interface, enviando e recebendo mensagens SOAP. Pode ser utilizado em conjunto com o HTTPRIO. |
| SOAPConnection | É utilizado para a conexão da aplicação com WebServices. |
| OpToSOAPDOMConvert | É utilizado para efetuar um parsercom relação aos métodos de chamada. Pode ser instanciado pelo objeto HTTPRIO. |
| HTTPSoapDispatcher | É utilizado nas chamadas para responder as solicitações com base no SOAP, de acordo suas configurações. |
| HTTPSoapPascalInvoker | É utilizado para a execução de determinado método de acordo com interpretação de uma mensagem SOAP. |
| WSDLHTMLPublish | É utilizado para a publicação de um documento WSDL para fornecer as informações do WebService. |

**Tabela 2.** Componetes do Delphi que podem ser utilizados em WebServices

necessárias do mesmo de acordo com o WebService: URL, UTF-8 (Cabeçalho) e SoapAction. Apropriedade SoapAction permite informar o propósito da requisição SOAP. Após o preenchimento das propriedades, o objeto ReqResp é executado enviando a requisição e armazenando a resposta na variável do tipo TMemoryStream, denominada "Stream".

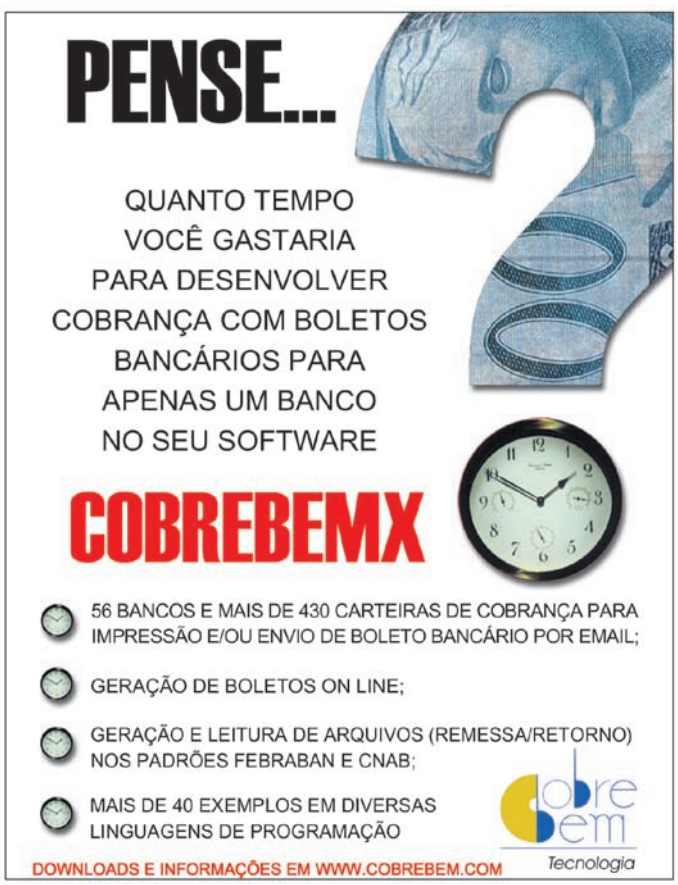
**Listagem 2.** Exemplo de consulta em WebService com HTTPReqResp

```
procedure TForm1.ButtonClimaHTTPReqRespClick(Sender: TObject);
var
  msoap, resposta: String;
  posDel: Integer;
  Stream: TMemoryStream;
  StreamStr: TStringStream;
  ReqResp: THTTPReqResp;
  XMLRetorno: TXMLDocument;
begin
  Stream := TMemoryStream.Create;
  msoap := '<?xml version="1.0" encoding="utf-8"?>'+
       '<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'+
       ' xmlns:xsd="http://www.w3.org/2001/XMLSchema"'+
       ' xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">'+
          '<soap:Body>'+
             '<GetWeather xmlns="http://www.webserviceX.NET">'+
                '<CityName>'+ EditCidade.Text +'</CityName>'+
                '<CountryName>'+ EditPais.Text +'</CountryName>'+
             '</GetWeather>'+
          '</soap:Body>'+
       '</soap:Envelope>';
  ReqResp := THTTPReqResp.Create(Form1);
  ReqResp.URL := 'http://www.webservicex.net/globalweather.asmx';
  ReqResp.UseUTF8InHeader := True;
  ReqResp.SoapAction := 'http://www.webserviceX.NET/GetWeather';
try
    ReqResp.Execute(msoap, Stream);
    StreamStr := TStringStream.Create('');
    StreamStr.CopyFrom(Stream, 0);
    resposta := StreamStr.DataString;
    posDel := 0;
    posDel := pos('Temperature', resposta);
    if posDel <> 0 then
      Delete(resposta, 1, posDel +15);
    posDel := pos('lt;/', resposta);
    if posDel <> 0 then
      Delete(resposta, posDel, Length(resposta));
    ShowMessage('Temperatura: '+ resposta);
    finally
      FreeAndNil(StreamStr);
      FreeAndNil(Stream);
      FreeAndNil(ReqResp);
    end;
end;
```

Variáveis deste tipo são utilizadas para o armazenamento de dados em memória dinâmica, permitindo assim que os mesmos sejam trabalhados na memória e posteriormente possam ser obtidos ou gravados em arquivos por exemplo. Também écriada uma variável do tipo TStringStream, que foi denominada "StreamStr", para copiar o valor da variável Stream, possibilitando assim a obtenção do valor em String no trecho de código "resposta := StreamStr.DataString". No próximo passo, a variável denominada "posDel" recebe o resultado da função Pos(). A função Pos() é utilizada para obter a posição da primeira ocorrência de determinado texto (no caso "Temperature", o elemento que contém o valor da temperatura), retornando assima posição do primeiro caractere. Posteriormente é utilizado o procedimento Delete() do Delphi para apagar o conteúdo da variável resposta que não será utilizado, de acordo com a posição obtida com a função Pos(). Estas funções são utilizadas neste exemplo com o objetivo "filtrar a informação", obtendo assim somente o valor desejado que é a temperatura, diretamente através da String. Para concluir, a temperatura é exibida e os objetos são liberados da memória.

## HTTPRIO

O Objeto HTTPRIO é um componente do Delphi que permite a conversão das chamadas aos métodos de uma interface em requisições baseadas em SOAP. Ele utiliza um HTTPReqResp para a conexão com o WebService. No momento em que a aplicação efetua um "Cast" do HTTPRIO para uma interface registrada, é gerada uma tabela de métodos na memória, disponibilizando assim uma implementação para tal interface. Sendo assim, o HTTPRIO executa tais métodos codificando as chamadas em requisições SOAP e envia uma solicitação para o WebService. Concluído o processo, ele obtém a mensagem de resposta da solicitação e adquire os parâmetros de saída e o valor de retorno da requisição. Vale à pena evidenciar também que assim como o HTTPReqResp, as propriedades do HTTPRIO devem ser configuradas de acordo com o WebService que será utilizado para que o mesmo possa ser executado.

A **Tabela 2** demonstra algumas propriedades importantes do HTTPRIO que podem ser configuradas de acordo com as informações do serviço oferecido.

Como é possível observar, com o preenchimento da propriedade WSDLLocation é possível efetuar uma configuração independente da linguagem utilizada no servidor, onde posteriormente as propriedades Service e Port também dever ser configuradas.

Criamos então um novo botão no formulário para fazer a consulta utilizando o componente HTTPRIO. Posteriormente, o menu File>New deverá ser acessado novamente, porém agora a opção Other deve ser utilizada, bem como em WebServices, selecionar a opção WSDL Importer. Este assistente pode ser utilizado toda vez que se deseja importar um documento WSDL que contém a descrição do WebService que será utilizado. Ele é muito útil, pois gera automaticamente todas as definições e interfaces que posteriormente poderão ser utilizadas através do objeto THTTPRio. Voltando a utilização do assistente, na opção "Location of..." deve ser definido o endereço do WebService, "http://www.webservicex.net/globalweather.asmx?wsdl", uma vez que este exemplo está usando um WebService do site www.webservicex.net para a consulta do clima. Posteriormente basta manter a opção Automatic SOAP versioning para a identificação do protocolo SOAP e finalizar o assistente. Como pode ser observado nesta etapa, o assistente gera uma Unit denominada globalweather.pas que pode ser gravada em conjunto com o projeto. Vale a pena evidenciar que o Delphi utiliza Interfaces do tipo Invokable na constituição da Unit. Uma Interface Inbokablepresente no Delphi permite que as interfaces de definição possam ser compiladas em tempo de execução com as informações. A **Listagem 3** demonstra um exemplo de código que foi inserido no segundo botão para efetuar a pesquisa no WebService utilizando a Unit globalweather.pas com base no documento WSDL e obter somente o valor da temperatura.

Como é possível observar neste exemplo, a função GetWeahter() obtida atravésda importação do documento WSDL do WebService, é utilizada para a consulta eobtenção do XML de retorno da mesma, tendo seu valor armazenado navariável "resposta". A variável "StreamStr" foi criada apenas como uma maneira depossibilitar a gravação do XML de retorno para o arquivo denominado "Clima.xml" no disco local, sendo um trecho de código opcional. Mais uma vez são utilizadas as funções Pos() e Delete() para filtrar a informação, obtendo assim somente o valor datemperatura.

A **Figura 1** demonstra o resultado final da aplicação, composta por ambas as consultas comentadas anteriormente.

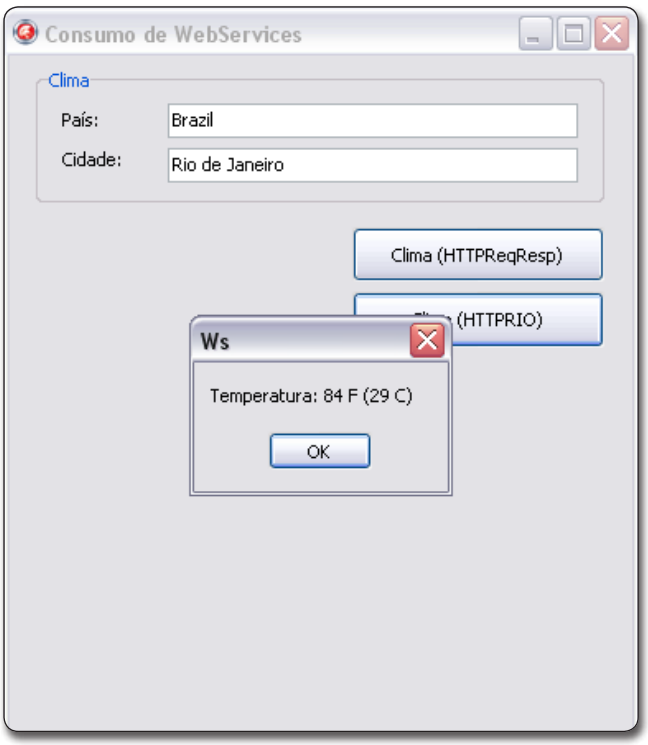



**Figura 1.** Resultado final da apliação WebService

**Listagem 3.** Exemplo de consulta em WebService com HTTPRIO

```
procedure TForm1.ButtonClimaHTTPRIOClick(Sender: TObject);
var
  resposta: String;
  posDel: Integer;
  XMLRetorno: TXMLDocument;
  StrStream: TStringStream;
begin
  resposta := GlobalWeatherSoap(GetGlobalWeatherSoap).GetWeather
  (EditCidade.Text, EditPais.Text);
  StrStream := TStringStream.Create('');
  StrStream.WriteString(resposta);
  StrStream.SaveToFile('C:\Clima.xml');
  posDel := 0;
  posDel := pos('<Temperature>', resposta);
  if posDel <> 0 then
      Delete(resposta, 1, posDel +13);
  posDel := pos('</Temperature>', resposta);
  if posDel <> 0 then
      Delete(resposta, posDel, Length(resposta));
  ShowMessage('Temperatura: '+ resposta);
end;
```

| Propriedade | Descrição |
|---|---|
| HTTPWebNode | Define várias propriedades para o acesso, como a configuração de um certificado e o Proxy. Este conjunto de propriedades é equivalente as propriedades do objeto HTTPReqResp. |
| URL | Especifica o endereço do WebService, normalmente constituído de [http:// ou https://] + [endereço]. |
| WSDLLocation | Propriedade que deve ser utilizada para definir o endereço do documento WSDL do WebService, permitindo assim uma procura de informações de uma maneira dinâmica durante a execução deste documento. Normalmente o endereço é constituído de [http:// ou https://] + [endereço]?wsdl. |
| Service | Após definida a propriedade WSDLLocation, pode ser definido facilmente a identificação do serviço de acordo com as informações carregadas. |
| Port | Após definida a propriedade WSDLLocation, pode ser facilmente definida a Porta do serviço. |

**Tabela 2.** Propriedades importantes do HTTPRIO

## Nota do DevMan

A utilização de Cast (Fundição) é muito comum nas linguagens orientadas a objetos. Na informática, o Cast é um processo utilizado para definir qual é o tipo do objeto que está em uso. Ele deve ser utilizado quando existe a atribuição de um objeto que pertence a uma classe superior da hierarquia das classes a um objeto que pertence a uma classe inferior.

## Outros recursos de Rede e obtenção de informações

Desde versões anteriores a XE 2, o Delphi já oferecia diversos objetos e ferramentas que possibilitavam a integração de aplicações com a Internet, simplesmente para a navegação (como a elaboração de uma aplicação WebBrowser), consumo de WebServices como foi comentado ou troca de informações entre as aplicações em uma rede. Também vale a pena evidenciar que o Delphi dispõe dos objetos TTCPServer e TTCPClient, que são encontrados na aba Internet da versão XE 2. O objeto TTCPServer permite adicionar a uma aplicação Delphi a possibilidade de prover funcionalidades através de uma rede. Tal objeto pode ser configurado com um determinado endereço IP e "ficar na escuta" de uma determinada porta, aguardando a recepção de alguma informação enviada por uma aplicação cliente para executar determinada tarefa, e em contrapartida, o objeto TTCPClient, inserido em uma outra aplicação Delphi, é o responsável por enviar tais solicitações.

No que diz respeito a acessar informações de páginas da internet com o Delphi, um objeto do tipo TWebBrowser também pode ser utilizado para a elaboração de uma aplicação robô, ou em outras palavras, uma aplicação capaz de preencher formulários de uma determinada página e obter valores das mesmas. Vale a pena evidenciar que ao utilizar aplicações robô direcionadas a efetuar preenchimentos de certas páginas, dependendo da elaboração ou necessidade do projeto, todas às vezes em existirem alterações consideráveis na estrutura do site, a aplicação robô poderá funcionar parcialmente ou em alguns casos até deixar de funcionar, sendo assim alterações nas páginas podem implicar em alterações na aplicação. O objeto TWebBrowser prove a funcionalidade de navegador Web a aplicação Delphi, permitindo assim a abertura e visualização de páginas da Internet.

Para dar início àcriação de uma aplicação robô, basta iniciar um novo projeto e inserir um objeto do tipo TWebBrowser no formulário. Este simples exemplo demonstra como efetuar uma busca no Google através de uma aplicação Delphi, efetuando o preenchimento do campo de pesquisa e posteriormente clicando no botão "Pesquisa Google". Com o WebService1 inserido, a propriedade Silent deve ser definida como True, fazendo com que não sejam exibidas mensagens de erros de scripts durante a abertura de uma página, que podem interromper a execução da aplicação. Feito isso, para iniciar a navegação em uma página, basta utilizar o procedimento "Navigate([http://] + [endereço])", onde o endereço que se deseja abrir deve ser passado como parâmetro. Contudo, para fazer com que a aplicação aguarde o carregamento da página e só então prossiga, a propriedade ReadyState pode ser comparada com "READYSTATE_COMPLETE" através de um laço, verificando assim o status da página para prosseguir. Com a página carregada, a propriedade OleObject do WebBroser1 pode ser utilizada para preencher os dados utilizando o identificador (ID) ou nome (Name) do elemento na página Web. Neste momento, o conhecimento da linguagem HTML (HyperTextMarkupLanguagem), a base padrão adotada nas páginas de internet, se torna muito importante. Por exemplo, para preencher determinado campo de um formulário presente em uma página da internet, deve-se informar o nome ou id do mesmo que pode ser obtido através do código fonte da página, assim como para efetuar um clique em determinado botão presente em um formulário da Web, existe a necessidade de se conhecer sua identificação. Contudo, comandos de teclado também podem ser utilizados para enviar o foco para os componentes presentes na página, como é o caso da tecla TAB. Para o envio de teclas para um objeto TWebBrowser, basta atribuir o foco ao mesmo via código (WebBrowser1.SetFocus) e utilizar a função Keybd_event(), da Unit Windows, que permite o envio de teclas diretamente através de código. Dando continuidade ao exemplo, foram adicionados mais 2Labels e 2 Edits para o preenchimento da URL do site e a palavra-chave que se deseja pesquisar. Também foi adicionado um Memo que exibirá o código HTML da página, assim como um botão para efetuar o processo. A **Figura 2** demonstra a aparência final da aplicação Robô.

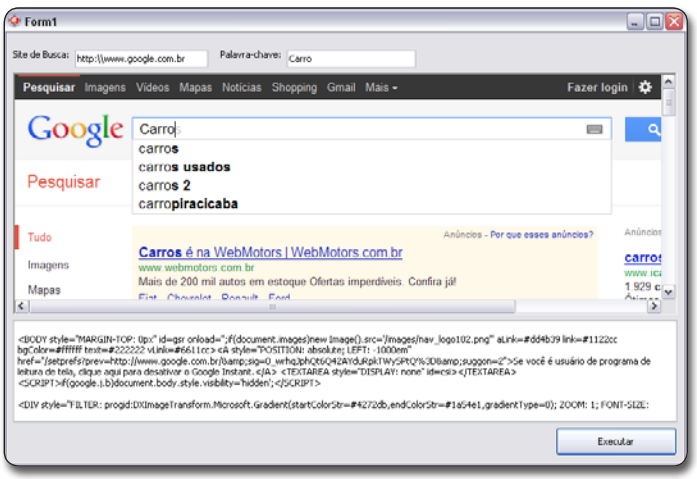



**Figura 2.** Aparência final da apliação Robô

A **Listagem 4** demonstra o código utilizado no evento OnClick do botão executar para iniciar a utilização da aplicação Robô.

Como pode ser observado no código da **Listagem 4**, o Web Browser1 é utilizado para navegar até página definida no Edit1. Text (http://www.google.com.br). Na próxima linha é executado um Loop while para permitir que o próximo código seja executado apenas quando a página estiver carregada, contudo foi utilizado o procedimento Sleep() para permitir uma visualização melhor dos passos, uma vez que tal procedimento faz com que o Delphi efetue "uma pausa" durante os milissegundos especificados. No próximo item é utilizada a função GetElementByID() e através da sua propriedade Value, é atribuído o valor da palavra-chave que se

```
<input class="lst lst-tbb" value="" title="Pesquisar" size="41" type="text" autocomplete="off" id="lst-ib" name="q" maxlength="2048"/>
```

**Figura 3.** Demonstração de como obter o id de um elemento

**Listagem 4.** Código do Evento OnClick do botão Executar

```
procedure TForm1.ButtonExecutarClick(Sender: TObject);
var
  DocHtml: Variant;
begin
  WebBrowser1.Silent := True;
  WebBrowser1.Navigate(Edit1.Text);
while Webbrowser1.ReadyState <> READYSTATE_COMPLETE do
    Application.ProcessMessages;
  Sleep(1000);
  WebBrowser1.OleObject.Document.GetElementByID('lst-ib').Value := Edit2.Text;
  Sleep(1000);
  WebBrowser1.OleObject.Document.all.Item('btnK', 0).Click;
  DocHtml := WebBrowser1.Document;
  Memo1.Lines.Text := DocHtml.Body.OuterHTML;
end;
```

deseja pesquisar. Como pode ser observado, a função é utilizada sobre um documento presente em um objeto OLE. Contudo, para obter o Id que deve ser informado (neste caso "lst-ib") é necessário analisar o código fonte do site Google. A **Figura 3** demonstra um exemplo de como identificar o Id do campo de pesquisa da página do Google.

Contudo, para facilitar a busca, normalmente os campos que podem ser preenchidos estão contidos em um formulário e utilizam a tag inicial "input", o que pode facilitar na hora de efetuar uma pesquisa para a obtenção do Id desejado. No próximo item da **Listagem 4**, é informado o nome do Item na qual se deseja clicar, neste caso o btnK. Para concluir, a variável DocHtml recebe neste caso o código fonte (HTML) do elemento Body e o mesmo é exibido através do objeto Memo1.

## Conclusão

Como pode ser observado neste artigo, a Internet oferece diversas possibilidades no que diz respeito a troca de informações entre as mais variadas aplicações e que podem ser facilmente aproveitadas por uma aplicação Delphi, utilizando determinados objetos como o HTTPReqResp e o HTTPRIO. Com a utilização de WebServices, as aplicações podem enviar mensagens baseadas no padrão SOAP para determinado serviço on-line e obter a reposta desejada, sem que exista a necessidade da intervenção de pessoas, lembrando que na Internet são disponibilizados os mais variados WebServices, que vão desde simples consultas de climas de uma região até consultas complexas, gratuitos, restritos ou com determinado custo.

Contudo, para utilizar os serviços disponibilizados na Internet pelos WebServices, ou até mesmo efetuar consultas em páginas da internet através de formulários e implementar o conceito de aplicações Robô, existe a necessidade do conhecimento dos objetos presentes no Delphi assim como o funcionamento do WebService ou Site na qual se deseja obter o recurso desejado.

**Giuliano Scombatti Pinto**
**giuliano@sygnux.com.br**
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

**Mais informações sobre WebServices**
*http://www.w3.org/TR/ws-arch/*

**DevSpace do Autor**
*http://www.devmedia.com.br/space.asp?id=283274*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
**www.devmedia.com.br/paycontent**

DevMedia

# Prism XE2, Windows P

## Utilizando controles Silverlight em aplic

## Resumo DevMan

***De que trata o artigo:***

A mais nova plataforma móvel da Microsoft, Windows Phone 7, vem se estabelecendo fortemente no mercado e na mesma medida já é há algum tempo uma realidade de negócios e trabalhos para os desenvolvedores. Atualmente, com o recente Prism XE2, a comunidade Delphi teve acesso também a este nicho, expandindo ainda mais suas possibilidades de projetos. Em vista disso, o artigo essencialmente dá sequência à introdução sobre o desenvolvimento de aplicações para Windows Phone apresentado anteriormente. Agora é explorada a efetiva definição das interfaces de usuário deste tipo de aplicações, que se dá através do uso de controles da tecnologia Silverlight.

***Em que situação o tema é útil:***

Se você é desenvolvedor Delphi e tem a pretensão de começar a estudar e eventualmente produzir aplicações para a plataforma Windows Phone 7 (WP7), deve ter em mente que além de suas funcionalidades, estes aplicativos móveis devem possuir uma boa estrutura e definição visual. Sendo assim, durante este aprendizado será imprescindível obter conhecimento sobre as nuances do Silverlight. Esta tecnologia, que ficou conhecida popularmente por ser voltada ao desenvolvimento de aplicações de internet ricas, as populares RIA (Rich Internet Applications), é a principal responsável pela definição das interfaces de usuário das aplicações WP7. Dito isto, você verá detalhes sobre a utilização dos controles Silverlight disponíveis à plataforma, bem como a essência da estruturação destes em uma aplicação.

***Prism XE2, Windows Phone e Silverlight:***

Ao mesmo tempo do estrondoso lançamento da versão XE2 do Delphi, a Embarcadero apresentou também o novo Delphi Prism, agora chamado simplesmente de Prism. Entre suas inúmeras novidades, não há como deixar de destacar seu estável e sólido relacionamento com o .NET Framework e, por conseguinte, o acesso a tudo de novo que a plataforma proporciona. Dentre estes itens novos, se encaixa o desenvolvimento para Windows Phone, que abre uma nova gama de possibilidades para a comunidade Delphi. Para iniciar no assunto, um quesito quase que obrigatório é a leitura do artigo sobre o tema publicado anteriormente nesta mesma revista. Na sequência, pode-se estipular também como necessário, um conhecimento prévio sobre Oxygene e Silverlight. Oxygene, como sabemos, é a linguagem adotada pelo Prism, que por também ser baseada em Pascal, é muito próxima da que estamos acostumados a utilizar no Delphi tradicional. Já o Silverlight é a tecnologia para RIA (Rich Internet Applications) da Microsoft, e que no Windows Phone 7 é utilizada principalmente para a definição da parte visual das aplicações, através do uso de seus controles disponíveis. Portanto, além da parte conceitual e obrigatória, será feita também uma abordagem prática demonstrando a utilização destes controles em aplicações reais. A partir disso, o leitor terá então um esboço de onde poderá chegar neste tipo de desenvolvimento, em função do uso do Silverlight em suas aplicações.

Ao se desenvolver aplicações para a plataforma Windows Phone 7, toda a definição da interface visual destas se dá através de XAML, que é uma linguagem de marcação baseada em XML, provida pelo Silverlight. Já de outras literaturas, o Silverlight é conhecido como uma tecnologia voltada para a plataforma Web, contudo, a Microsoft elaborou um subconjunto do mesmo, exclusivo para a

# hone e Silverlight

## ações Windows Phone no Prism XE2

Fabrício Hissao Kawata

plataforma Windows Phone. Assim, tornou-se disponível para uso sua gama de controles de UI (User Interface), na mesma proporção para o desenvolvimento Web, salva algumas pequenas exceções. Para se fazer menção a alguns exemplos, destacam-se os elementos mais tradicionais utilizados na construção de aplicativos de interação com o usuário, tais como Button, RadioButton, TextBlock, TextBox, entre outros.

É importante ressaltar que, para o ideal aprendizado sobre o desenvolvimento de aplicações para a plataforma Windows Phone, se faz necessária a introdução sobre a tecnologia Silverlight e sua forma peculiar de trabalhar com seus controles. Uma abordagem diferente disso seria pular etapas, o que não é o esperado em qualquer estudo.

### Controles Silverlight – Introdução

No IDE do Prism XE2 (Visual Studio), os controles Silverlight ficam dispostos na ToolBox, em uma categoria específica, conforme mostra a **Figura 1**. Logo, são inevitáveis os comparativos com os controles da VCL disponíveis na Tool Palette do Delphi tradicional, tanto no aspecto de funcionalidade como em seus detalhes de estrutura.

Uma justificativa para isso é que os controles disponibilizados pelo Framework Silverlight, assim como os controles da VCL, possuem algumas propriedades comuns entre si. Graças a uma sólida hierarquia de classes, acaba por se estabelecer uma padronização entre os controles, facilitando o aprendizado e otimizando o desenvolvimento. A classe Control é base da maioria dos controles Silverlight e provê grande parte de suas propriedades disponíveis. Dentre estas, algumas se destacam pela maior usabilidade no cotidiano da programação, tais como as que definem o tamanho, o nome, a visibilidade e a fonte de texto de determinado controle.

Elucidando este contexto na prática e ao mesmo tempo fazendo menção à tradicional VCL, pode-se citar algumas situações. Por exemplo, em um projeto Delphi Win32 é comum se estabelecer uma nomeação dos controles utilizados através de sua propriedade Name, para que cada elemento ganhe uma identificação condizente com sua função e facilite na codificação. Esta parte de código se justifica pelo fato de que, o acesso ao controle e, conseguintemente à suas demais propriedades, se darão a partir de seu nome definido. Sendo assim, um TEdit que servirá de caixa de entrada para a informação "nome do cliente", poderia ser nomeado como edtNomeCliente, facilitando sua identificação perante ao formulário (TForm) em que está inserido. No Silverlight esta mesma definição se dá pela propriedade x:Name, enquanto que o elemento similar ao TEdit é o TextBox. A seguir é demonstrada uma simples notação destes controles (VCL e Silverlight):

*VCL*

```
object edtNomeCliente: TEdit
    Text = 'edtNomeCliente'
end
```

*Silverlight*

```
<TextBox x:Name="txtNomeCliente" />
```

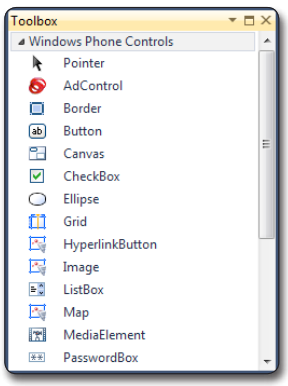



**Figura 1.** Controles disponíveis na Toolbox

Mantendo esta mesma linha de pensamento, conforme citado, há ainda outras propriedades triviais como altura, largura e visibilidade, somente para citar algumas. Para as duas primeiras, assim como na VCL, estas propriedades são as já tradicionais Height e Width, que são utilizadas para dimensionar visualmente o elemento. Já a questão da visibilidade é a forma de esconder ou mostrar determinado controle em tela. Para esta definição, este paralelismo que se faz com as aplicações Delphi tradicionais não se dá da mesma maneira. Isto porque enquanto na VCL tem-se a presença da propriedade Visible, que possui valor booleano (true ou false), em controles Silverlight a propriedade similar é chamada Visibility, cujo valor é um enumerado. Esta estrutura provê dois exclusivos valores possíveis: Visible e Collapsed. Novamente demonstrando em código o uso destas propriedades, haveria algo como:

*VCL*

```
object edtNomeCliente: TEdit
  Width = 121
  Height = 21
  Visible = False
end
```

***Silverlight***

```
<TextBox x:Name="txtNomeCliente"
         Width="121"
         Height="21"
         Visibility="Collapsed"/>
```

Deixando de lado a comparação com aplicações VCL, cabe mencionar que há ainda outras propriedades adicionais comuns exclusivas aos controles Silverlight, tais como IsEnabled, que permite definir se o controle está habilitado ou não para interação com o usuário, e Background, para personalizar o fundo do controle.

## Controles de layout

Os controles Silverlight podem ser divididos em pequenos grupos, em razão de suas funcionalidades. Dito isto, é importante frisar que o estudo de um destes grupos é vital para a utilização ideal dos demais em um projeto. O grupo em questão é o chamado "controles de layout" (layout controls) que, como o próprio nome sugere, provê a definição da disposição e posicionamento dos controles em determinada tela do projeto, uma vez que funcionam como uma espécie de container para estes controles. Por uma questão conceitual, na verdade, na estrutura de uma aplicação Silverlight existem telas, mas sim páginas (pages), da mesma maneira que em uma aplicação VCL Forms tem formulários. Outro detalhe é que todas as páginas XAML que um projeto WP7 venha a ter sempre herdará direta ou indiretamente da classe PhoneApplicationPage.

Os principais controles de layout disponíveis para a plataforma WP7 são: Canvas, Grid, StackPanel, Border e ScrollViewer. Este último, aliás, é de grande utilização em aplicativos móveis, pois devido ao tamanho da tela do dispositivo, seu espaço vertical para conteúdo é bastante reduzido se comparado a uma tela Desktop, por exemplo. Dessa forma, se você necessita desenvolver um aplicativo com um formulário de entrada de dados muito longo, que extrapole as dimensões do aparelho, uma boa pedida é envolver todo o conteúdo em um controle ScrollViewer. Na prática este scroll fica transparente, sendo percebido somente pelo ato de "rolar" a tela. A **Figura 2** faz uma amostragem do uso deste controle mostrando os dois momentos da UI, onde o primeiro demonstra a aparência inicial do formulário e o segundo após a rolagem efetuada na tela do dispositivo. Sua sintaxe é extremamente simples e a prova disso é o trecho a seguir, responsável por definir a UI da figura anteriormente mostrada:

```
<ScrollViewer Height="600" HorizontalAlignment="Left"
        Margin="12,6,0,0" Name="scrollViewer1" VerticalAlignment="Top" Width="438">
  <StackPanel Orientation="Vertical">
    <TextBlock Height="30" HorizontalAlignment="Left"
          Margin="12,6,0,0" Text="Código" VerticalAlignment="Top" />
    <TextBox Height="71" Text="Informe seu código" Width="460" />
    <TextBlock Height="30" HorizontalAlignment="Left"
          Margin="12,6,0,0" Text="Nome" VerticalAlignment="Top" />
    <TextBox Height="71" Text="Informe seu nome completo" Width="460" />
    <TextBlock Height="30" HorizontalAlignment="Left"
          Margin="12,6,0,0" Text="Data Nascimento" VerticalAlignment="Top" />
    <TextBox Height="71" Text="Informe sua data nascimento" Width="460" />
    ...
    <Button Content="Enviar"></Button>
  </StackPanel>
</ScrollViewer>
```

**Figura 2.** Utilização do controle de layout ScrollViewer

Canvas pode ser definido como o controle de layout mais simples de todos, uma vez que ele exerce a função de um container básico para os demais controles. Ele é útil quando se deseja obter uma liberdade maior quanto ao posicionamento dos controles em tela. Este, aliás, é o diferencial do Canvas, que define o posicionamento de seus controles internos através de coordenadas X e Y, fazendo uso de um mecanismo denominado attached properties (propriedades anexadas). Uma propriedade anexada possibilita que determinado controle tenha acesso às propriedades de seu elemento pai. Vendo isto na prática, o código a seguir mostra a utilização de um controle de layout Canvas, contendo outros dois controles TextBox, que são então posicionados em tela através dos valores definidos para as propriedades Top e Left de seu elemento pai Canvas.

```
<Canvas x:Name="ExemploCanvas">
  <TextBox Canvas.Top="20" Canvas.Left="20" Width="300" />
  <TextBox Canvas.Top="80" Canvas.Left="20" Width="300" />
</Canvas>
```

Outro layout control muito utilizado em aplicações Siverlight é o StackPanel que, como seu próprio nome sugere, faz menção a algo relacionado a "pilhas". Isto porque o StackPanel faz o empilhamento dos controles em faixas horizontais e verticais. É ideal para montagens rápidas de layout nestas orientações. Um exemplo disso já foi visto anteriormente no uso do ScrollViewer, no qual foi utilizado um controle StackPanel para manter os controles alinhados verticalmente na tela do dispositivo.

Em termos de flexibilidade e organização, o controle de layout Grid é a opção, uma vez que trabalha de forma semelhante a uma tabela (table) HTML. Em outras palavras, sua estrutura é definida por linhas e colunas onde estarão dispostos os controles internos. Para compreender melhor seu funcionamento, nada melhor que visualizar isto em código. Portanto, a seguir é mostrado um trecho

XAML que define um Grid de dimensões 4x4 (quatro linhas e quatro colunas) e que pode ser visto na **Figura 3**:

```
<Grid x:Name="ExemploGrid" ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
</Grid>
```

**Figura 3.** Controle de layout - Grid

Pelo código, fica exposto que a definição quanto ao número de linhas e colunas se dá dentro das seções RowDefinitions e ColumnDefinitions, respectivamente. Como complemento, a inserção dos controles nesta estrutura se dá por meio do uso de attached properties, conforme visto anteriormente com Canvas. Dessa forma se, hipoteticamente, fosse necessário posicionar elementos na primeira linha do Grid, sendo estes um controle TextBox na primeira coluna e um Button na segunda coluna. Suas definições se dariam internamente à estrutura <Grid>, da seguinte forma:

```
<Grid x:Name="ExemploGrid" ShowGridLines="True">
...
  <TextBox Grid.Row="0" Grid.Column="0" />
  <Button Grid.Row="0" Grid.Column="1" Content="Button 1" />
</Grid>
```

Sem deixar de mencionar, há ainda o controle de layout Border, que se trata de um container simples para controles, porém, com um adicional decorativo. Isto porque, com ele é possível se definir uma borda em torno de sua área e, consequentemente, em torno da área que envolve os controles. É possível inclusive "envolver" outro controle de layout, conforme mostra código XAML a seguir, onde está sendo reutilizado o exemplo Canvas mostrado anteriormente:

```
<Border BorderBrush="Red" BorderThickness="10">
  <Canvas x:Name="MainCanvas" >
    <TextBox Canvas.Top="20" Canvas.Left="20" Width="300" />
    <TextBox Canvas.Top="80" Canvas.Left="20" Width="300" />
  </Canvas>
</Border>
```

## Controles de interação

Uma vez vista a forma de se dispor os controles Silverlight numa aplicação Windows Phone, através do uso dos controles de layout, o próximo passo do aprendizado é obter um pouco mais de conhecimento sobre os controles visuais do Framework. Controles estes que serão os usados nos aplicativos, para que haja a ideal interação com o usuário. Traçando novamente um paralelo com a VCL, em Win32 haveria como exemplo TButton, TEdit, TLabel, TImage, TMemo, entre outros.

Tomando como base os controles já utilizados nos exemplos anteriores, tem-se como inicial os elementos TextBox e Button. O primeiro representa a tradicional caixa de textos, muito utilizada em formulários de entrada de dados. Dessa forma, quando o usuário interage com o aplicativo, fornecendo informações através da digitação dos dados, comumente estes precisam ser submetidos para outro local, seja um arquivo, um banco de dados ou mesmo a própria memória do dispositivo. Para que isso ocorra, é comum direcionar esta ação para controles como Button, que em aplicações para a plataforma Windows Phone ganham uma conotação ainda maior. Isto porque aplicativos móveis necessitam ser enxutos o suficiente para compensar o tamanho reduzido disponível, e sendo assim, cada controle deve ter sua função muito bem definida no contexto. O controle Button, se comparado a um TButton da VCL, possui uma peculiaridade interessante, referente ao seu texto de exibição. Enquanto em Win32 o botão possui uma propriedade denominada Caption, que recebe uma String simples, no Silverlight existe uma propriedade Content, em que é definido um texto ou até mesmo um outro controle, abrindo uma gama de possibilidades. Num exemplo mais tradicional, podemos definir então um controle Image como sendo o conteúdo de um botão, conforme mostra a **Figura 4**. A seguir, é exibido o código XAML responsável por esta definição:

```
<Button Width="200" Height="100">
  <Button.Content>
    <Image Source="Imagem.png" />
  </Button.Content>
</Button>
```

Outra necessidade comum em aplicativos, independente do segmento, é a apresentação de textos estáticos. Tais textos são apenas dizeres que não possuem nenhuma interação dinâmica,

mas proveem mais uma forma de informação ao usuário. Como exemplo, podemos citar um trecho de "recomendações" ao usuário, uma seção de direitos reservados da aplicação, dentre outros. Para estas situações o Silverlight provê o controle TextBlock que, como o próprio nome sugere, é um bloco de texto. Justamente por isso sua propriedade Text é a mais importante, conforme pode ser visto em sua declaração XAML a seguir:

```
<TextBlock Text="Texto Texto Texto Texto Texto Texto"
      FontFamily="Arial" FontSize="30">
</TextBlock>
```

**Figura 4.** Botão cujo conteúdo é outro controle (Image)

## Controles de listas

Dentre os controles de interação com o usuário, apresentam-se aqueles cuja função básica é manipular listas de dados. Prova disso é o controle RadioButton provido pelo Silverlight, que proporciona uma maneira rápida e fácil de apresentar ao usuário uma lista de opções possíveis sobre determinado contexto. Supondo uma aplicação WP7 para a editora DevMedia, um exemplo poderia apresentar ao assinante uma forma de escolha sobre sua revista preferida, num sistema de votação, conforme esboçado na **Figura 5**.

O código XAML necessário para a definição da lista de opções é apresentado a seguir:

```
<StackPanel Orientation="Vertical">
 <TextBlock Text="Qual sua revista preferida?" FontSize="28"></TextBlock>
 <RadioButton x:Name="rdbClubeDelphi" GroupName="Magazines"
       IsChecked="True" Content="Clube Delphi" />
 <RadioButton x:Name="rdbJavaMagazine" GroupName=" Magazines "
       IsChecked="False" Content="Java Magazine" />
 <RadioButton x:Name="rdbNetMagazine" GroupName=" Magazines "
       IsChecked="False" Content=".Net Magazine" />
 <RadioButton x:Name="rdbMobileMagazine" GroupName=" Magazines "
       IsChecked="False" Content="Mobile Magazine" />
 <RadioButton x:Name="rdbSQLMagazine" GroupName=" Magazines "
       IsChecked="False" Content="SQL Magazine" />
 <RadioButton x:Name="rdbEngenhariaSoftware" GroupName=" Magazines "
       IsChecked="False" Content="Engenharia de Software" />
</StackPanel>
```

Notam-se neste trecho as principais propriedades de um elemento Radio Button, tais como Content, IsChecked e GroupName. A primeira, Content, é utilizada para definir o texto a ser exibido ao lado do item de opção. Num grupo de opções é viável que se queira definir um item como sendo o default entre as escolhas, ou seja, aquele que irá começar selecionado. Para ações deste tipo define-se então a propriedade IsChecked, que determina a seleção de um item. Uma particularidade do controle Radio Button é que ele permite que o usuário somente selecione uma opção por vez, não importando o número de elementos disponíveis em tela. Isto poderia causar algum inconveniente, caso fossem disponibilizadas opções com contextos diferentes (Ex: um grupo de opções para a escolha da revista preferida e outro grupo sobre sua cor predileta). Já prevendo este tipo de situação, o controle oferece a propriedade GroupName, onde define-se um nome comum com o intuito de se "agrupar" opções. Dessa forma, elementos Radio Button que possuam o mesmo GroupName formarão um grupo. Cada grupo poderá então ter somente um único elemento selecionado, que é o que acontece com o trecho de código apresentado, onde todos os elementos pertencem ao grupo Magazines.

**Figura 5.** Uso do controle RadioButton

Todavia, o controle Silverlight que evidencia mais fortemente uma lista é o ListBox, que exibe uma listagem de itens verticalmente em tela. Tais itens podem ser definidos um a um, via XAML, ou providos de qualquer tipo coleção. A primeira situação é a mais trivial e um exemplo de sua declaração pode ser vista a seguir:

```
<ListBox x:Name="MagazinesList">
  <ListBox.Items>
    <ListBoxItem Content="Clube Delphi" IsSelected="True" />
    <ListBoxItem Content="Java Magazine" />
    <ListBoxItem Content=".Net Magazine" />
    <ListBoxItem Content="Mobile Magazine" />
    <ListBoxItem Content="SQL Magazine" />
    <ListBoxItem Content="Engenharia de Software" />
  </ListBox.Items>
</ListBox>
```

O ListBox ainda provê uma propriedade ItemsSource que, a grosso modo, recebe a fonte de dados que formará a lista de itens do controle. Um exemplo desta "fonte" seria uma coleção qualquer, seja de strings, objetos, etc. Este tipo de definição é feita via code-behind, utilizando-se tal propriedade. Numa situação prática,

aproveitando a mesma ideia da listagem de revistas DevMedia, pode-se considerar a seguinte lista:

```
var revistas: List<System.String> := new List<System.String>();
revistas.Add ('Clube Delphi');
revistas.Add ('Java Magazine');
revistas.Add ('.Net Magazine');
revistas.Add ('Mobile Magazine');
revistas.Add ('SQL Magazine');
revistas.Add ('Engenharia de Software');
```

De posse desta coleção, para usá-la como fonte de itens para o List Box de revistas anteriormente mostrado (MagazinesList), bastaria a seguinte definição, utilizando sua propriedade ItemsSource:

```
MagazinesList.ItemsSource := revistas;
```

O resultado final de ambas as abordagens pode ser vista na **Figura 6**.

**Figura 6.** Uso do controle ListBox

## Controles de multimídia

Dentro do grupo de controles Silverlight, há aqueles destinados para o trabalho com arquivos multimídia, tais como imagens, músicas e vídeos. Tais controles enriquecem ainda mais o poder das aplicações para a plataforma WP7, uma vez que seguem a tendência atual deste nicho.

Da mesma forma que se exibe um simples texto estático, o controle Image, já citado anteriormente, possibilita a exibição de uma imagem estática em tela, similarmente ao TImage da VCL. Sua utilização ganha mais importância em razão do apelo visual cabível às aplicações móveis. Em termos de suporte, o controle trabalha com as já tradicionais extensões .jpg e .png, proporcionando as mesmas possibilidades de outras plataformas. Já em sua sintaxe, basicamente deve-se definir duas de suas propriedades: Source e Stretch. A primeira refere-se ao caminho da imagem a ser exibida, enquanto que a segunda define a forma com a imagem será disposta visualmente. Os valores possíveis para Strech são: Fill, None, Uniform e UniformToFill. Cada valor deste muda o comportamento do controle em relação à exibição da imagem. Por exemplo, quando se atribui o valor None, a imagem é mostrada em seu tamanho normal, ao mesmo tempo em que Fill estende a imagem por toda a área disponível no controle, podendo ocasionar distorções na mesma. UniformFill age da mesma forma que Fill, porém o aspecto original da imagem é preservado. As definições das propriedades Source e Strech podem ser vistas no trecho de código a seguir:

```
<Image Source="DevMedia.png"
    Stretch="None" >
</Image>
```

Se o aplicativo que está sendo desenvolvido necessita ou tem a pretensão de incorporar e reproduzir vídeos ou músicas, o Silverlight possui o controle MediaElement. Na própria marketplace, facilmente podemos encontrar aplicativos para Windows Phone que nitidamente fazem uso deste tipo de controle, tais como aplicações de notícias, players diversos e jogos, somente para citar alguns. Um detalhe que não pode deixar de ser mencionado quando se trabalha com o controle MediaElement, é prover para a aplicação o suporte à mudança de orientação, ou seja, poder visualizar o vídeo em modo paisagem. Isto é necessário para que o usuário possa facilmente maximizar o espaço disponível em tela para o vídeo. Para habilitar este recurso na aplicação, basta alterar os atributos SupportedOrientation e Orientation da MainPage do projeto para Landscape. Em termos práticos, a declaração básica de um controle MediaElement em XAML se dá como a seguir, onde podem ser vistas suas duas propriedades principais: Source e AutoPlay. Source indica a localização do vídeo que, no caso dos dispositivos móveis, geralmente é informada a URL do mesmo. Isso se explica pelo fato de ser inviável a distribuição de vídeos juntamente dos aplicativos, sendo assim, comum a hospedagem do vídeo em um servidor, acessando-o pela aplicação através da internet. AutoPlay indica se o vídeo deverá ser tocado automaticamente ou se será necessária a intervenção do usuário. Essas propriedades podem ser vistas no trecho a seguir:

```
<MediaElement AutoPlay="True"
        Source="http://www.site.com/videos/video.wmv"/>
```

## Definindo ações

Até agora foram vistos alguns exemplos dos controles Silver light disponíveis para uso em aplicações da plataforma Windows Phone, no que diz respeito à interação com o usuário. No entanto, assim como ocorre em projetos Win32, não basta somente "decorar" a tela somente com controles deste tipo. Em determinado momento, para que realmente se estabeleça uma interação e o usuário possa fazer uso de tais controles, será imprescindível a

codificação de seus eventos relacionados, tais como o clique de um botão, a seleção de um item de uma lista, o passar de mouse sobre um TextBox, entre outros. Devido a isto, cada arquivo XAML de um projeto no Prism para a plataforma WP7 possui um arquivo .pas correspondente, denominado code-behind. A grosso modo, este arquivo tem a mesma função de um arquivo .pas de um projeto VCL, uma vez que é nele que estará o código-fonte de uma determinada página (Page) XAML. Para esta codificação, utiliza-se então a linguagem Oxygene que, por também ser baseada em Object Pascal, facilita bastante o cenário para o desenvolvedor Delphi, diminuindo drasticamente sua curva de aprendizado.

Na visão prática, para exemplificar melhor a manipulação de eventos de um controle Silverlight, tomemos como ponto de partida uma ação trivial e clássica, que é a definição do evento Click de um controle qualquer. Mais especificamente, supondo que este controle se trate de um simples Button, a definição de seu evento Click em Silverlight pode se dar de duas formas: no XAML ou no code-behind. A primeira e mais usual, se dá ainda na declaração na própria declaração XAML do controle:

```
<Button Content="Executar Ação"
    Click="button1_Click"/>
```

Ao se definir um nome ao método Click, tenha em mente que este será o nome a ser buscado no code-behind, portanto, é imprescindível que este exista. Com base da declaração XAML, no .pas da página teríamos o evento da seguinte forma:

```
method MainPage.button1_Click(sender: System.Object; e: System.Windows.Routed
EventArgs);
begin
 //Codificação
end;
```

Uma vez definido o manipulador de eventos do botão, nele poderá ser incluso qualquer codificação necessária. Continuando, a outra forma de definir um manipulador de eventos é diretamente no code-behind, comumente já no momento da inicialização da página. Um exemplo seria usar para isso um evento Loaded que, como o próprio nome sugere, ocorre quando a página já está carregada, com todos os seus controles já desenhados em tela. Neste momento então se atribui ao Click do objeto Button um novo manipulador de evento ButtonControl_Click. Da mesma forma que anteriormente, será necessário que este evento já esteja declarado no código.

```
method MainPage.LayoutRoot_Loaded(sender: System.Object; e: System.Windows.
RoutedEventArgs);
begin
 button1.Click += new RoutedEventHandler(button1_Click);
end;
```

## Indo um pouco além: interação com formas

Os tipos de controles vistos até aqui podem ser considerados os essenciais para a criação de aplicações para a plataforma Windows Phone 7 para fins distintos. No entanto no Silverlight, quando se trata da interação com o usuário, além dos controles, não se pode deixar de mencionar o seu trabalho com formas de objetos. Tais objetos também são considerados controles, tais como Ellipse, Rectangle e Polygon, que serão os demonstrados nesta abordagem.

O controle Ellipse, conforme o próprio nome sugere, cria um objeto círculo na tela. Para sua definição básica há o envolvimento de duas propriedades principais, Width e Height, que determinam respectivamente sua largura e altura. Assim sendo, para se obter um círculo perfeito, bastaria informar um valor idêntico a essas duas propriedades. Uma vez definido o tamanho do objeto, outra prática comum é poder se definir as cores do mesmo, tais como a de seu preenchimento e contorno. A cor de preenchimento é determinada pela propriedade Fill do controle, enquanto que a cor de contorno é definida por Stroke. Portanto, uma elipse amarela e com contorno azul, poderia ser declarada da seguinte forma no Silverlight:

```
<Ellipse Width="300"
     Height="300"
     Stroke="Blue" StrokeThickness="10"
     Fill="Yellow"/>
```

A menção a ser feita em relação a este simples trecho de código fica por conta da atribuição de uma cor sólida a qualquer propriedade de um objeto Silverlight. Neste caso em específico, vemos que a propriedade Fill do controle Ellipse recebe um simples valor do tipo string ("Yellow"), para representar a cor amarela. Para que este valor seja devidamente convertido para uma cor efetiva, ainda em tempo de execução, o próprio Silverlight a converte para um objeto SolidColorBrush, que recebe como argumento a referida cor através de um enumerado de System.Colors, tal como SolidColorBrush(Colors.Yellow). Tal enumerado disponibiliza um conjunto padrão de cores de sistema, ou seja, as cores básicas que estão disponíveis num dispositivo Windows Phone.

Seguindo a ordem de exemplos de "formas" de objetos tem-se o controle Rectangle que, obviamente, provê um retângulo em tela. Sua utilização inicial é praticamente idêntica ao do controle Ellipse, conforme pode ser visto a seguir:

```
<Rectangle Width="300"
      Height="100"
      Stroke="Blue" StrokeThickness="10"
      Fill="Yellow"
      RadiusX="30"
      RadiusY="30"/>
```

Logo, nota-se a utilização das mesmas propriedades vistas anteriormente. Em razão da presença dos valores para a definição da altura e largura do controle, pode-se obter facilmente um quadrado perfeito, bastando fornecer valores idênticos para Width e Height. Como complemento, foram exibidas duas outras propriedades, RadiusX e Y, que proporciona um incremento visual ao controle, fazendo com que seus cantos fiquem arredondados.

Por fim, citando o último exemplo de controle nesta análise prévia do trabalho com formas de objetos diversos com Silverlight, há o polígono (Polygon). Este controle, como sua própria nomeação indica, desenha em tela uma figura geométrica de forma livre. Em função disso, ao contrário de elementos com formas definidas, como os vistos anteriormente, não há a definição de propriedades como Width e Height, que informam sua largura e altura. Com o Polygon usa-se, obrigatoriamente, um conjunto de pontos que determinarão a sua formação:

```
<Polygon Stroke="Blue" StrokeThickness="10"
    Fill="Yellow"
    Points="8,8,90,200,300,8" />
```

O resultado de todas as formas apresentadas pode ser vista na **Figura 7**.

**Figura 7.** Ellipse, Rectangle e Polygon

## Estilizando controles

Uma página HTML é formada por vários elementos que compõem sua estrutura e que resultam na página a ser vista pelo usuário final. Cada elemento deste, na mesma medida que os controles Silverlight vistos até aqui, possuem propriedades que definem sua aparência visual. No HTML utiliza-se então uma linguagem de estilo denominada CSS (Cascading Style Sheets) como forma de centralizar o controle da aparência dos mais variados controles de um WebSite. O "estilo" de um elemento é então separado do elemento em si, alcançando assim um nível de abstração ideal. Dessa forma, uma das vantagens proporcionadas por esta prática é a definição e padronização de estilos para grupos de elementos e a facilidade em se alterar em um único lugar a aparência de vários elementos ao mesmo tempo.

Aproximando este contexto ao do Silverlight, pode-se tomar como base os exemplos de controles mostrados neste artigo. Tomando como base o controle Ellipse suas propriedades dizem respeito às suas cores de preenchimento e contorno (Fill e Stroke), que poderiam perfeitamente serem "extraídas" para um local central, e assim estar disponível a futuros controles deste tipo. Isto será útil para que neste momento seja demonstrado como é possível, no Silverlight, criar um estilo para tais controles, utilizando-se da mesma ideia do uso de CSS no HTML. Passando esta situação para o plano real, suponha que o aplicativo desenvolvido tenha uma fundamentação padrão de cores, onde todos os controles possuam a mesma definição de valor para uma mesma propriedade. Determinado cliente então solicita a mudança deste padrão. Numa situação tradicional, sem uso de estilos, seria necessário percorrer a aplicação toda, controle a controle, fazendo a devida mudança, gerando retrabalhos, além da já duplicação de código XAML a cada controle.

Para se definir um estilo de controle no Silverlight, a codificação básica envolvida é mostrada a seguir:

```
<Style x:Key="Name" TargetType="Type">
  <Setter Property... />
  <Setter Property… />
  ...
</Style>
```

Pelo código, fica nítida a utilização de um elemento Style, cujas principais propriedades são: x:Key e TargetType. O primeiro é o identificador do estilo, servindo para referenciá-lo no código e diferenciá-lo de outros estilos que forem criados. Já TargetType determina o tipo de controle ao qual é destinado o estilo. Um exemplo clássico seria a definição de um estilo exclusivo aos botões da aplicação. Neste caso, TargetType teria o valor "Button".

Além das propriedades, internamente à estrutura de Style há várias definições Setter que, efetivamente, definem os valores (Value) das propriedades (Property) do tipo de controle ao qual o estilo é destinado. Dessa forma, para se definir uma cor de fundo azul para controles Button, teríamos uma definição Setter como a seguir:

```
<Setter Property="Background" Value="Blue" />
```

Vale ressaltar que é perfeitamente possível se declarar um estilo sem uma TargetType definida. Isso quer dizer que o estilo não será específico a um determinado tipo de controle. No entanto, nestes casos, deve-se assegurar que todas as propriedades de controles definidas em Setter sejam condizentes com o controle em que se está aplicando o estilo. Do contrário, erros ocorrerão. Esta mesma teoria também é válida para estilos com TargetType explícita.

Uma vez definido o elemento Style, o estilo já pode ser aplicado nos controles desejados. Para isto, ainda em sua declaração XAML, utiliza-se sua propriedade Style para esta atribuição. A seguir, tem-se a sintaxe básica a ser utilizada, onde NomeDoEstilo é o valor definido para a propriedade x:Key do elemento Style:

```
<Controle Style="{StaticResource NomeDoEstilo}" />
```

Partindo para a exemplificação prática, iremos então definir um estilo para o controle Ellipse mostrado anteriormente.

O local onde esta definição deverá ser feita depende do contexto onde será aplicado o estilo. Isso se explica pelo fato de que um estilo pode ser definido localmente, servindo assim somente àquela página ou globalmente (na seção Resources do arquivo App.xaml), ficando disponível à aplicação como um todo.

Utilizando a primeira abordagem (local) e supondo que os controles do exemplo estivessem num mesmo StackPanel, a definição do estilo se daria dentro da seção Resources do controle de layout:

```
<Style x:Key="DevMediaStyle" TargetType="Ellipse">
  <Setter Property="Fill" Value="Yellow" />
  <Setter Property="Stroke" Value="Blue" />
  <Setter Property="StrokeThickness" Value="10" />
</Style >
```

Esta mesma declaração de Style poderia ser feita na seção Resources da aplicação, ficando assim disponível globalmente:

```
<Application.Resources>
  <Style x:Key="DevMediaStyle" TargetType="Ellipse">
  <Setter Property="Fill" Value="Yellow" />
  <Setter Property="Stroke" Value="Blue" />
  <Setter Property="StrokeThickness" Value="10" />
 </Style >
</Application.Resources>
```

Finalizando, para a utilização deste estilo nos controles de formas que estão sendo utilizados como exemplo, teria-se um código XAML da seguinte forma:

```
<Ellipse Style="{StaticResource DevMediaStyle}"
     Width="300"
     Height="300"/>
```

## Conclusão

Foi visto neste artigo uma abordagem mais prática sobre o desenvolvimento de aplicações para a plataforma móvel Windows Phone 7, através do Prism XE2. Dessa forma, o intuito inicial deste trabalho foi de servir como um complemento imediato ao artigo anterior sobre o assunto, publicado nesta revista, onde foi evidenciado um início de estudo prezando pela definição sólida dos conceitos básicos envolvidos neste tipo de desenvolvimento. Mais especificamente, foram vistos os primeiros passos para a definição das interfaces de usuário dos aplicativos para WP7, onde são utilizados os controles Silverlight, exclusivos para a plataforma. Esta definição visual ganha ainda mais importância em vista do contexto em que se está trabalhando (plataforma móvel).

Mantendo esta mesma dinâmica, como consequência, em futuros artigos pretendo manter esta evolução, partindo para tópicos mais avançados, tais como a integração com Web Services e serviços Cloud.

Uma prática que já é tradicional e corriqueira em artigos da revista Clube Delphi é o de deixar desafios aos leitores, relacionados ao assunto apresentado, incentivando assim a continuidade do estudo e posterior crescimento profissional. Portanto, seguindo esta linha, minha dica (ou desafio) é pelo estudo aprofundado do assunto – Windows Phone e Prism XE2 – partindo dos textos publicados nesta e em outras edições, passando por trabalho com dados e interação com o próprio Hardware. Dessa forma, o aumento do número de interessados no assunto certamente resultará em mais feedbacks pela rede (fóruns, blogs, sites e comunidades) e mais fontes de informação para profissionais de todos os níveis.

Agradeço a todos e até a próxima.

**Fabrício Hissao Kawata**
*fabricio.kawata@bol.com.br*
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas.
Blog: prismsilver.blogspot.com

**Embarcadero Prism XE2 Enterprise - Trial**
*https://downloads.embarcadero.com/free/prism*

**Microsoft Silverlight**
*http://www.silverlight.net*

**Windows Phone SDK 7.1**
*http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=27570*

**Dê seu feedback sobre esta edição!**

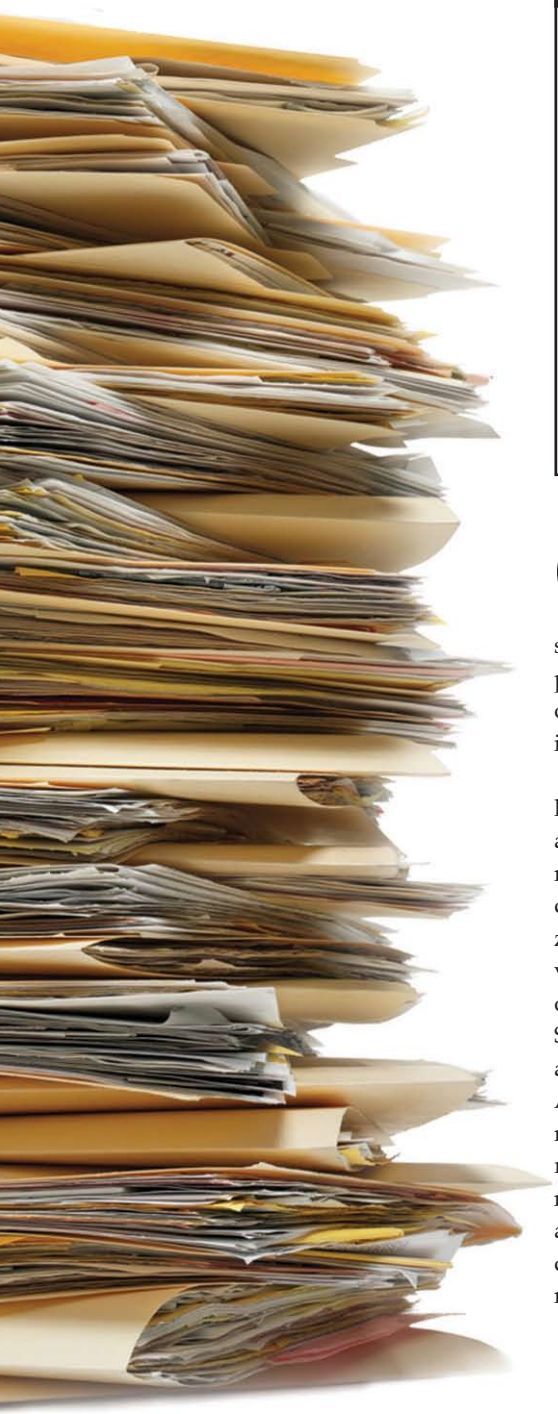
A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Desenvolvimento NoSQL no Delphi XE 2

Paulo Quicoli



## Resumo DevMan

***De que trata o artigo:***

Sobre entender o que é um banco de dados NoSQL, suas características, aplicações e como é possível utilizar essa tecnologia com Delphi.

***Em que situação o tema é útil:***

Conhecendo os recursos e aplicações do NoSQL é possível criar aplicações híbridas, aliviando assim alguma possível sobrecarga em bancos de dados relacionais, direcionando isso aos bancos NoSQL.

***Desenvolvimento NoSQL no Delphi XE 2:***

Mais uma sigla surge no mercado, a NoSQL. Como desenvolvedores não podemos ficar fora disso, é preciso entender do que se trata e se é possível utilizar com Delphi. O objetivo deste artigo é esclarecer de forma simples, prática, consciente o que são os bancos de dados NoSQL e implementar o uso de um deles, o mongoDB. Através de um exemplo vamos centralizar operações de log em um banco NoSQLe mostrar como é possível utilizá-lo com o Delphi em operações simples.

Quando somos chamados para desenvolver um novo projeto, logo pensamos em como esses dados serão armazenados, consequentemente já pensamos em qual banco de dados utilizar, qual estrutura de tabelas empregar, logo, imaginamos um banco relacional SQL.

Os bancos relacionais, juntamente com a linguagem SQL, têm controlado a área de armazenamento de informações por pelo menos 30 anos. Não havia, e ainda não há, dúvidas de que são o padrão para armazenagem de informações complexas provenientes de sistemas de negócio. O fato é que não existia alternativa à altura da dupla SGBDR e SQL. Seu uso tem sido justificado até hoje por sua propriedade transacional ACID, de atomicidade, consistência, isolamento e durabilidade, o que garante segurança aos sistemas de negócio. A evolução natural dessa tecnologia criou sistemas de alta disponibilidade, recursos como replicação, suporte a dados estruturados e muito mais. Isso tudo tem um custo. Bancos de dados relacionais, no geral, não são muito eficientes no escalonamento horizontal, sua especialidade é a vertical. Ou seja, se seu sistema demanda mais processamento do banco de dados, melhora-se o servidor, incluindo mais memória, mais espaço físico, mais processadores, etc.

Grandes empresas, que possuem produtos que gerenciam centenas de milhões de usuários, como Google, Facebook, Amazon e Twitter, não utilizam por trás de seus sistemas um banco relacional, utilizam algo especificado como NoSQL.

## Um novo paradigma

NoSQL é uma definição para uma forma de armazenamento de dados que pode variar dependendo de quem a explica. Pode ser explicada como "Sem SQL" ou "Não apenas SQL". Essas são definições que não expressam suas características de armazenamento. Em sua ideia básica, um banco de dados NoSQL pode ser representado pelo acrônimo BASE:

• Basicamente, disponível: a disponibilidade é a marca de um banco NoSQL. Através do uso de mecanismos de replicação e o uso de sharding (particionamento de dados por diferentes servidores) resultam em um sistema de informações onde falhas são parciais, onde pequenas porções de toda informação armazenada fica indisponível por curtos períodos de tempo. Assim, pode-se dizer que é um banco de dados sempre disponível;
• Estado flexível: enquanto os sistemas relacionais garantem consistência de informações, sistemas NoSQL permitem a inconsistência de dados. Isso acontece porque no mundo NoSQL não existe a necessidade de se definir um esquema rígido para os dados, permitindo que suas informações se adequem rapidamente às mudanças de negócio envolvidas, deixando contudo, ao desenvolvedor a tarefa de lidar com isso;
• Eventualmente consistente: Embora os aplicativos precisam lidar com uma consistência instantânea, o NoSQL garante que em algum ponto adiante no tempo a informação estará consistente. Isso é bastante contrário ao pensamento ACID, onde o commit de uma transação assegura consistência.

Essas são características básicas, mas existem implementações que asseguram, por exemplo, o conceito de atomicidade em suas operações de atualização/inserção de dados. Mas toda essa diferença vai um pouco além. Quando utilizamos um banco de dados relacional pensamos em como armazenar os dados, porém, o cenário atual nos leva a pensar diferente, nos leva a perguntar "como utilizamos os dados". O armazenamento agora é projetado com base em como usamos as informações.

Atualmente temos a criação de informação acontecendo de forma muito dinâmica, gerando volumes de dados que levam as grandes empresas a espalhá-los por servidores ou datacenters quando um único servidor (ou datacenter) não dá mais conta. E essas informações precisam estar disponíveis, mesmo com queda de algum desses datacenters. Este é um cenário enfrentado por negócios envolvidos com redes sociais, onde essa conexão social dita funcionalidades tão facilmente quanto as apaga, inclusive, sem pensar nelas antes. Isso demanda que os dados não estejam presos a um esquema rígido como os dos bancos relacionais, que mesmo podendo ser alterados, causariam uma rigidez e perda de agilidade na resposta de novas demandas.

## Funcionamento

A regra máxima para o armazenamento de dados em um banco NoSQL é agrupar as informações necessárias em um único registro. Por exemplo, imaginando o conceito Pessoa, que possui um endereço, vários telefones e vários e-mails. No mundo relacional essas informações estariam distribuídas em tabelas, com seus devidos relacionamentos. Já no NoSQL, toda essa informação estaria disponível em um único registro. Isso é possível, pois não se emprega a teoria relacional neles, mas outras técnicas que caracterizam os tipos de bancos NoSQL. Temos como tipos:
• Key/ValueStore: armazena suas informações em estruturas de dados semelhantes a dicionários, formando o par chave/valor. Este tipo de estrutura é a mais simples, porém é a que mais possui capacidade de armazenamento e escalabilidade. Os seguintes bancos utilizam essa estratégia: Berkeley DB (pertencente à Oracle), TokyoCabinet, MemcacheDB e outros;
• WideColumnsStore: sua estrutura permite o conceito de subcolunas, além de milhares de colunas e linhas. Seus representantes são: BigTable (pertencente à Google), Cassandra (Apache), HiperTable e outros;
• DocumentStore: este tipo de banco de dados representa um conceito (como a pessoa citada anteriormente) como um documento a ser armazenado. Esse documento contento as informações do "registro" é formatado em documentos XML ou JSON e são localizador por um identificador único. RavenDB, MongoDB e CouchDB são exemplos;

**Nota**

Esses são os tipos mais difundidos, existem outros mais complexos.

## mongoDB

É um banco NoSQL open source que foi escrito em C++. Suas principais características são o armazenamento de dados no formato de documentos (JSON/BSON), que é fácil de lidar e gerenciar, uma vez que não exige a criação de esquemas; Suporte a índices que, como em qualquer tipo de armazenamento de dados, agilizam o resultado de pesquisas, você pode pensar nesses índices da mesma forma como você pensaria nos índices de um banco relacional; Replicação, que pode ser assíncrona entre servidores, para garantir a disponibilidade através de LANs e WANs; Sharding para escalonamento horizontal, o detalhe é que sua implementação permite uma arquitetura de auto particionamento, onde o próprio mecanismo automaticamente realiza o balanceamento de carga e distribuição das informações; Pesquisas dinâmicas através de sua linguagem própria de consulta, que é muito similar ao SQL; Atualização atômica de informações; Implementação de um conceito chamado Map/Reduce, que aperfeiçoa operações em lote e operações onde agregação de elementos é envolvida, a grosso modo é como uma operação de filtro para essas situações; Armazenamento de arquivos binários.

Todos esses recursos chamaram a atenção de empresas como Disney e Foursquare, que o utilizam em seus produtos.

**Nota do DevMan**

BSON é a abreviação para Binary JSON, que é um a representação binária de um documento JSON serializado. É leve, ou seja, sua representação possui apenas as informações necessárias mais uma sobrecarga mínima para que funcione e pode ser utilizado nos mais diversos sistemas e ambientes.

## Cenários de uso

Devido à sua conceituação que difere do modelo relacional, é preciso identificar os cenários onde realmente o mongoDB pode

fazer a diferença. Um desses cenários seria a centralização de logs. Imagine que você tem desenvolvido para uma determinada empresa vários sistemas, cada um armazenando seu log individualmente. Caso seja necessário identificar as operações feitas por um determinado usuário, seria necessário procurar em cada sistema o que foi feito. Seria muito mais simples para seu cliente final possuir um único local onde todas as ações estariam armazenadas e que fossem fáceis de serem localizadas. Esse é um dos cenários, a base de dados real estaria livre de armazenamento desse tipo de informação. Outro cenário é se você desenvolve aplicativos para gerenciamento de conteúdo, por não possuir a necessidade de geração de esquemas, fica fácil criar novas funcionalidades que impactariam na estrutura de dados. O mongoDB também é bem aceito em aplicativos de ECommerce quando utilizado como uma ferramenta de apoio aos bancos de dados relacionais existentes. Nessas situações ele poderia cuidar do controle dos "carrinhos", listas de produtos, e deixar o fechamento de pedidos para o relacional.

Contudo existem alguns cenários não indicados como, sistemas que precisam de transações complexas ou sistemas que utilizam data warehousing para análises que não sejam em tempo real.

## O ambiente

O servidor do mongoDB é representado pelo aplicativo mongod.exe, localizado na pasta bin de sua instalação (uma fez feito o download e tendo descompactado oarquivo). Esse serviço pode ser configurado para ser executado como um serviço do Windows. Como já foi dito, apesar de não haver a necessidade da criação de esquemas (como tabelas, linhas ou colunas), existe um aplicativo que funciona como um shell para o mongoDB. Este shell é representado pelo aplicativo mongo.exe. É um aplicativo de linha de comando desenvolvido em JavaScript que é útil para inspecionar o conteúdo de um banco de dados, testas pesquisar, criar índices, executar scripts de manutenção e qualquer outra função administrativa. Para utilizar o shell é necessário que primeiramente o servidor (mongod.exe) esteja em execução, pois o shell automaticamente se conecta ao servidor e já utiliza um banco de dados chamado test.

## Operações básicas

Antes de utilizamos o mongoDB com o Delphi, vamos nos familiarizar com sua API e utilizar seu shell para realização de ações simples. Por padrão, o servidor mongoDB assume o caminho "C:\data\db" para criação de bancos de dados, essas duas pastas devem existir. Certificando-se que as pastas mencionadas existem, inicie o servidor efetuando um clique duplo sobre o mesmo, e inicie o shell, também efetuando um clique duplo. A **Figura 1** exibe o shell em execução.

Como o shell automaticamente se conecta a um banco chamado test, podemos inserir objetos nele. Para criar um objeto que possui a propriedade "Nome", simplesmente utilizamos o seguinte comando dentro do shell:

```
obj = {Nome: "Paulo"};
```

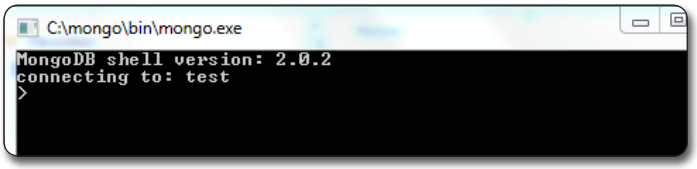



**Figura 1.** Shell do mongoDB

O comando cria um documento, a representação de um objeto, com as propriedades determinadas seguindo a notação JSON. O JSON (JavaScriptObjectNotation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos que é largamente utilizada por sua notação ser extremamente mais simples do que a XML. Não existem tags diversas nem regras complexas para escrita e muito menos mecanismos complexos de parser. Para se ter uma ideia observe a **Figura 2**, ela mostra a sintaxe para notação JSON para um objeto.

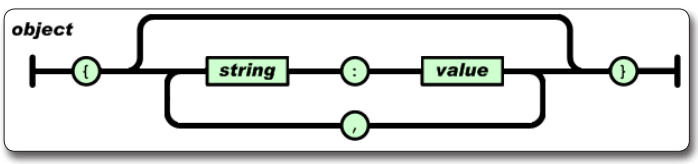



**Figura 2.** Representação JSON

Esse documento ficou armazenado na varíavelobj, que precisa ser persistida. Assim, o seguinte comando salva o objeto no banco de dados test, já existente:

```
db.test.save(obj);
```

O comando db aponta para um determinado banco de dados, que também é chamado de coleção. Uma vez acessando a coleção temos outros comandos que são executados contra a coleção indicada. No caso, o save irá salvar o objeto obj na coleção test. Veja que nenhum esquema de tabelas/campos foi definido. Uma vez salvo podemos agora fazer uma pesquisa geral. Para isso, temos o comando find que retorna todos os objetos da coleção:

```
db.test.find();
```

A **Figura 3** mostra o resultado. É importante observar que o mongo incluiu automaticamente um novo campo no objeto, o campo _id. Esse campo é gerado pelo mongo para representar a identificação única do objeto em uma coleção. O valor gerado de forma automática é conhecido como ObjectId. Sua formação é a seguinte:

4 bytes que representam data e hora, 3 bytes representando um id para a máquina, 2 indicando o id do processo, e 3 finais que funcionam como um contador. Essa data e hora segue o estilo unix, é um valor inteiro representando o número de segundos antes ou depois de 1º de Janeiro de 1970. O id da máquina é obtido dos três primeiros bytes do hash do nome da máquina ou, do endereço de rede, ou da máquina virtual (o que estiver em uso).

O id do processo na mais é do que o PID (identificador) do processo que originou o objeto. Por fim, o incremente é número aleatório sempre crescente. Por mais que o mongo cuide dessa tarefa, a recomendação é que se nós tivermos uma propriedade no objeto que sirva como chave candidata, devemos utilizá-la. Também é possível implementar uma geração sequencial de números inteiros (ver seção links).

**Figura 3.** Localização de um objeto

Ao invés de localizar todos os objetos contidos, também é possível especificar filtros para a pesquisa. Para testar, vamos inserir um segundo objeto, mas com propriedades diferentes, adicionamos agora uma propriedade Idade (**Figura 4**).

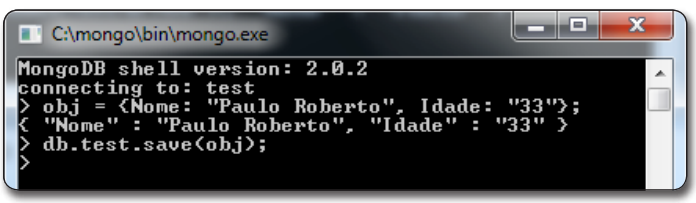



**Figura 4.** Salvando outro objeto

Vamos tentar localizar todos os objetos cuja propriedade Nome seja igual a "Paulo", e que seja listado o resultado:

```
db.test.find({Nome: "Paulo"}).forEach(printjson);
```

Operadores como >=, <= ,< e >são utilizados como a seguir:

```
db.test.find({"Nome": { $gt "Paulo"}});  Nome > "Paulo"
db.test.find({"Nome": { $lt "Paulo"}});  Nome < "Paulo"
db.test.find({"Nome": { $gte "Paulo"}});  Nome >= "Paulo"
db.test.find({"Nome": { $lte "Paulo"}});  Nome > "Paulo"
```

Esses operadores também podem ser utilizados para formarem limites como:

```
db.test.find({"Nome": { $gt "Paulo", $lt "Paulo Roberto }});  Nome > "Paulo" e <= "Paulo Roberto"
```

Assim, a **Figura 5** mostra o resultado da pesquisa: Objetos onde a propriedade Nome seja maior ou igual a "P".

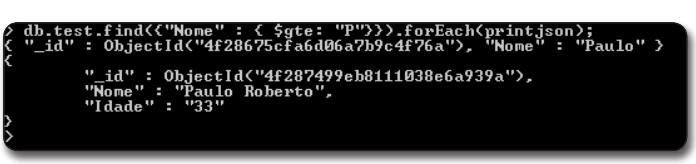



**Figura 5.** Resultado de pesquisa

Observe no resultado que o mongo retornou todos os objetos que obedeceram a condição, mesmo que eles tenham estruturas diferentes. Também é possível limitar a quantidade de objetos a serem retornados simplesmente adicionando o complemento limit(N), onde N é quantidade de objetos a ser retornada. Dessa forma, a pesquisa anterior poderia ser reescrita da seguinte forma para que apenas 1 objeto fosse retornado:

```
db.test.find({"Nome": {$gte: "P"}}).limit(1);
```

Para realizar a exclusão de um objeto utilizamos o comando remove. Este comando recebe como parâmetro uma condição, onde objetos que satisfazerem essa condição serão removidos:

```
db.test.remove({Nome: "Paulo"});
```

A atualização de um objeto pode ser feita utilizando o comando update, que recebe como parâmetro um filtro de localização e operadores de atualização. Vamos observar o seguinte comando:

```
db.test.update({_id: "1"}, {$set: {Nome: "Novo nome"}});
```

Como primeiro parâmetro foi informado uma condição para a aplicação do update, ou seja, apenas objetos com _id = 1 serão atualizados, na sequência utilizamos o operador $set para ajustar o valor de uma propriedade. É possível passar uma lista de propriedades que devem ter seu valor atualizado. Existe também um conjunto de operadores, cada um com uma funcionalidade específica a ser utilizado no momento de um update, por exemplo, $inc, $unset, $push, $pop, $rename e outros.

## Integrando com Delphi

Até o momento só vimos operações realizadas dentro do shell do mongo. Para que possa ser utilizado externamente com outras linguagens de programação, o mongo disponibiliza o conceito de drives, que são implementações de cada linguagem para especificações publicadas pelo mongoDB, assim, existem drives para as mais diversas linguagens, como Java, C#, C++, PHP, ColdFusion, Python, Ruby, Delphi, e outras.

Em Delphi, esse drive é representado pelo projeto TMongoWire, que mapeia o mongoDB para variáveis Delphi do tipo OleVariant para que a experiência de uso se assemelhe ao do JavaScript, contudo, ainda está em desenvolvimento.

Após a realização do download e descompactação do pacote do TMongoWire, criamos um aplicativo no Delphi onde vamos fazer uso do mongoDB para armazenamento de logs.

## Aplicativo de exemplo

É necessário adicionar ao projeto as seguintes units do TMongo Wire: mongoWire.pas (implementação da comunicação com o mongoDB), mongoID.pas(geração dos Ids no formato especificado pelo mongoDB) e bsonDoc.pas (implementação da serialização no formato BSON). O aplicativo possui apenas um formulário

(**Figura 6**) que possui uma divisão em guias realizada com um TPageControl.

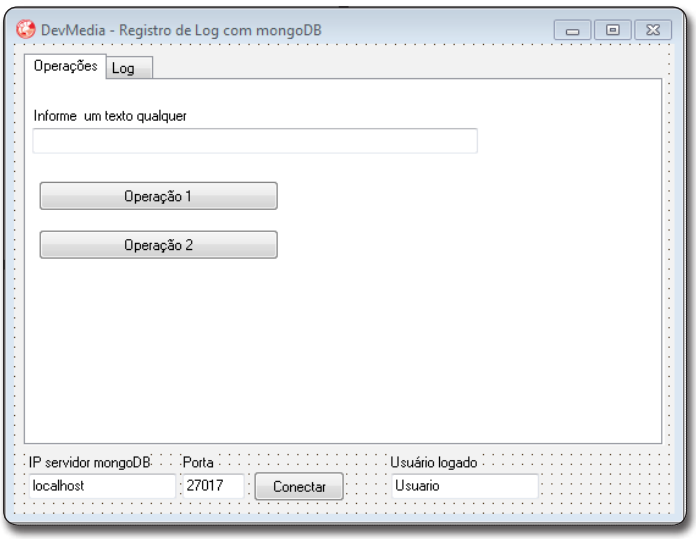



**Figura 6.** Formulário principal, guia operações

Na primeira guia temos as operações que serão realizadas e que serão armazenadas em log. No formulário em si também temos os parâmetros para comunicação com o servidor mongoDB. Por padrão, ao executar o servidor mongod.exe o mesmo espera conexões na porta 27017, porém isso pode ser alterado (consulte documentação). Já na segunda guia (**Figura 7**) vemos um ListView que conterá os logs armazenados no mongoDB, com opções para exclusão e consulta.

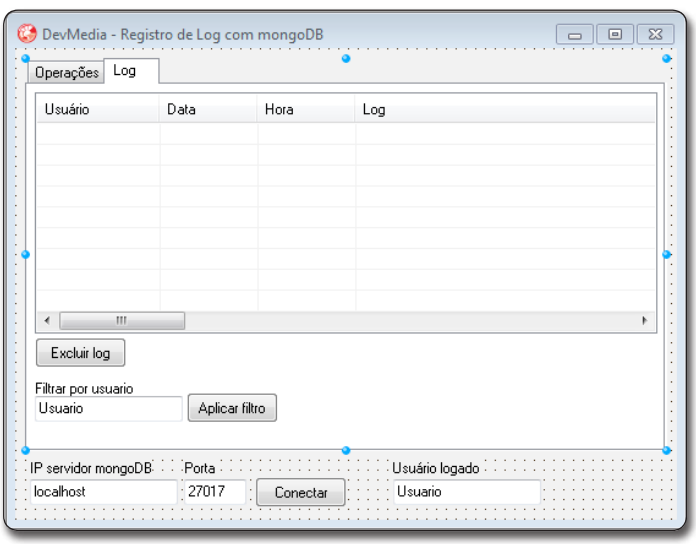



**Figura 7.** Formulário principal, guia Log

No formulário temos um campo privado do tipo TMongoWire que é instanciado no momento de criação do formulário e é destruído no momento em que o forme também é destruído. A classe TMongoWire disponibiliza métodos de comunicação com o mongoDB, assim, no evento onClick do botão conectar, tentamos realizar uma conexão através do método TMongoWire.Open.

Existe um detalhe na implementação da unitmongoID que é preciso corrigir. Segundo o modelo de geração de Ids apresentado anteriormente, parte da estrutura do Id é composta pelo número de identificação da máquina. No MongoWire isso foi implementado efetuando uma leitura na seguinte chave do registro: 'HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\MachineGuid', contudo, no ambiente utilizado por mim (Win7 64bits) o código utilizado não consegue efetuar a leitura dessa chave, assim, a função InitMongoObjectID foi alterada como é mostrado a seguir:

```
procedure InitMongoObjectID;
var
 r:TRegistry;
 g:TGUID;
begin
 r:=TRegistry.Create;
try
  //use cryptography machineguid, keep a local copy of this in initialization?
  r.RootKey:=HKEY_LOCAL_MACHINE;
  if r.OpenKeyReadOnly('\Software\Microsoft\Cryptography') then
     //g:=StringToGUID('{'+r.ReadString('MachineGuid')+'}')
      g:=StringToGUID('{D9CE36B0-2B8C-4010-9C52-474D739FBBF8}')
  else
     CreateGUID(g);//?
  mongoObjectID_MachineID:=g.D1 shr 8;
  mongoObjectID_Counter:=GetTickCount;//0?
  finally
     r.Free;
  end;
end;
```

Ao invés de obter o GUID a partir do registro, simplesmente gerei um (Ctrl+Shit+G) e o passei como parâmetro.

O aplicativo vai registrar o log contendo o nome do usuário, data, hora e operação realizada. Como o mongoDB trabalha com armazenamento de dados no formato de documentos BSON, o TMongoWire disponibiliza um método Insert que recebe como parâmetro um objeto do tipo IBSONDocument, que está definido na classe bsonDoc. Um banco NoSQL possui um conceito diferente de armazenamento, esses objetos que são inseridos, ficam alojados em estruturas chamadas simplesmente de coleções. Dessa forma nosso exemplo declarou uma constante que armazena o nome da coleção que vamos utilizar por todo aplicativo:

```
const
 colecao = 'logOperacoes.Registros';
```

No método TMongoWire.Insert também é preciso informar o nome da coleção que irá conter o objeto. Quando o primeiro objeto é inserido na coleção, o mongoDB então cria seu arquivo físico.

Centralizamos a utilização desse método no procedimento RegistrarLog, visto na **Listagem 1**.

A função BSON instancia um objeto do tipo TBSONDocument que implementa IBSONDocument. Como parâmetro passamos o conteúdo do documento, que segue a notação JSON já mostrada anteriormente. É facilmente possível identificar os atributos, como id, usuário, data, hora e log. Todos esses recebem seus

valores baseados nos dados lançados na tela, com exceção do id que recebe seu valor pela função mongoObjectID, esta obtém seu valor no formato especificado pelo mongoDB.

Nos evento onClick dos botões Operação 1 e Operação 2 chamamos RegistrarLog, inclusive no evento OnChange do TEdit. Dessa forma estamos simulando as ações que um sistema real pode possuir.

Uma vez que temos informações registradas, foi centralizado em método chamado CarregarLog a recuperação dessas informações. Na **Listagem 2** é possível ver a utilização da classe TMongoWireQuery.

**Listagem 1.** RegistrarLog

```
procedure TForm1.RegistrarLog(Texto: string);
var
 d: IBSONDocument;
begin
 d:=BSON([
    'id',mongoObjectID,
    'usuario',editUsuario.Text,
    'data',FormatDateTime('dd/MM/yyyy',Now),
    'hora',FormatDateTime('hh:mm:ss',Now), 'log',Texto
  ]);
 FMongoWire.Insert(colecao,d);
end;
```

**Listagem 2**. Carregando o Log

```
procedure TForm1.CarregarLog;
var
 q:TMongoWireQuery;
 d:IBSONDocument;
begin
 lv1.Items.BeginUpdate;
 try
    lv1.Items.Clear;
    d:=BSON;
    q:=TMongoWireQuery.Create(FMongoWire);
 try
    q.Query(colecao,nil);
    while q.Next(d) do LoadItem(lv1.Items.Add,d);
    finally
    q.Free;
 end;
 finally
    lv1.Items.EndUpdate;
 end;
end;
```

Ao instanciar um objeto TMongoWireQuery passamos em seu construtor a conexão em si, representada pela instância FMongoWire. TMongoWireQuery representa uma consulta no banco de documentos, assim é fornecido o método Query onde passamos como parâmetro a coleção que contém os dados a serem buscados e um segundo parâmentro que é o filtro da busca. Esse filtro também é especificado no formato BSON, mas se passamos nil em seu lugar, todos os documentos contidos na coleção são retornados. Para percorrer esses documentos é utilizado o método Next que transfere para uma variável do tipo IBSONDocument o documento corrente e retorna True enquanto existe um próximo documento a ser extraído.

Para cada documento que extraímos, o passamos como parâmetro no método LoadItem que simplesmente preenche o ListView (**Listagem 3**).

**Listagem 3**. Preenchendo o ListView

```
procedure TForm1.LoadItem(li: TListItem; d: IBSONDocument);
begin
 li.Caption:=VarToStr(d['usuario']);
 li.SubItems.Clear;
 li.SubItems.Add(VarToStr(d['data']));
 li.SubItems.Add(VarToStr(d['hora']));
 li.SubItems.Add(VarToStr(d['log']));
 li.SubItems.Add(VarToStr(d['id']));
end;
```

Como mencionado, o TMongoWire mapeia o tipo BSON para variants do Delphi, assim, para acessar os valores contidos em um objeto IBSONDocument utilizamos as funções de Variants de acordo com o tipo do dado a recuperar. Como nossos atributos são somente strings, foi utilizado somente a função VarToStr. Utilizamos o método CarregarLog logo após a conexão com o banco de documentos, já o método LoadItem é também utilizando dentro de RegistrarLog, logo após inserir o documento no banco, para que o mesmo já aparece no ListView.

Para realizar uma exclusão temos o método TMongoWire.Delete. Este recebe como parâmetro um filtro formatado também em BSON. Assim podemos especificar quais documentos remover. O código a seguir mostra como excluímos um documento que foi selecionado no ListView:

```
FMongoWire.Delete(colecao,BSON(['id',lv1.Selected.SubItems[3]]));
```

Informamos a coleção e um id. Como o id é único, apenas documento com o id especificado é removido. Se informássemos a propriedade data, todos documentos com a data informada seriam removidos.

A pesquisa de documentos, ou filtro, é muito semelhante ao método CarregarLog, com a diferença que passamos um documento no formato BSON indicando o filtro, como mostra a **Listagem 4.**

**Listagem 4**. Filtrando resultados

```
procedure TForm1.btnAplicarClick(Sender: TObject);
var
 q:TMongoWireQuery;
 d:IBSONDocument;
qry: IBSONDocument;
begin
 qry := nil;
 if editBusca.Text <>'' then
 begin
    qry:=BSON(['usuario',editBusca.Text]);
 end;
 lv1.Items.BeginUpdate;
 try
    lv1.Items.Clear;
    d:=BSON;
    q:=TMongoWireQuery.Create(FMongoWire);
 try
      q.Query(colecao,qry);
 while q.Next(d) do LoadItem(lv1.Items.Add,d);
 finally
      q.Free;
 end;
 finally
    lv1.Items.EndUpdate;
 end;
end;
```

## Conclusão

O NoSQL é uma tendência de tecnologia que veio para ficar. Para quem está acostumado com o mundo relacional, aplicar uma modelagem para esse tipo de banco de dados é uma mudança de pensamento e tanto. Por isso sua aplicação deve ser bem planejada. Montamos aqui um exemplo onde seu uso específico para armazenamento de logs, apesar de um pouco limitado com Delphi, pode trazer alguns benefícios como remover do banco de dados relacional principal informações de log que quase nunca são utilizadas. Um fato a ser considerado também é que, o NoSQL não veio para substituir os bancos relacionais, mas para completá-los, em alguns casos específicos seu uso é o mais adequado, como no Google, Facebook, Amazon, Twitter, mas em outros, continuaremos com bancos relacionais. Cabe a você desenvolver testar, analisar, e verificar em que caso sua aplicação é melhor ou não.

**Paulo Quicoli**
*pauloquicoli@gmail.com*
Editor da revista ClubeDelphi e editor técnico .Net Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas e coordenador técnico da equipe de desenvolvimento .NET da Siplan Control-M (www.siplancontrolm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga. Blog: http://quicoli.worpress.com


**mongoDB**
*http://www.mongodb.org/*

**TMongoWire**
*https://github.com/stijnsanders/TMongoWire*

**Lista de bancos NoSQL**
*http://nosql-database.org*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Design Patterns
## Utilizando os padrões Abstract Fac

## Resumo DevMan

***De que trata o artigo:***

Este artigo discute o uso de padrões de projeto (design patterns) em soluções de software, apresentando para isto cenários possíveis de utilização dos patterns Abstract Factory, Iterator e Observer.

***Em que situação o tema é útil:***

Design patterns são técnicas pré-definidas e empregadas na resolução de problemas em atividades que envolvam a construção de softwares. O pattern AbstractFactory representa uma alternativa bastante útil para a geração de instâncias de diversos tipos de objetos que estejam relacionados de alguma forma. Já Iterator permite a navegação sequencial ao longo de coleções de objetos, ao passo que o padrão Observer torna possível que objetos relacionados a uma determinada estrutura sejam notificados quando de mudanças de estado ocorridas nesta última.

***Design Patterns - Utilizando os padrões Abstract Factory, Observer e Iterator:***

Ao longo de diversos projetos de software será bastante comum que os desenvolvedores se deparem com a repetição de determinadas situações. Em situações como esta pode ser possível que já existam soluções para certos tipos de necessidades: ao conjunto de recomendações que representam uma alternativa para a resolução um problema específico dá-se o nome de padrão de projeto/design pattern. Padrões são o resultado direto de experiências bem sucedidas durante a elaboração de sistemas, sendo que sua aplicação incentiva não apenas o uso de boas práticas, como também pode conduzir a uma melhor organização estrutural dos softwares que se baseiam nos mesmos.


Em edições anteriores foi dado início a uma discussão sobre o uso de design patterns no desenvolvimento de software, levando em conta ainda para isto de conceitos de orientação a objetos. Este artigo tem por meta dar prosseguimento a esta abordagem, apresentando desta vez características e cenários envolvendo os padrões Abstract Factory, Iterator e Observer. Além disso, serão construídas três aplicações de exemplo, sendo que cada uma destas visa demonstrar como se utilizar estes patterns.

Design patterns (termo conhecido também em português como "padrões de projetos") são práticas voltadas à resolução de problemas específicos e que podem, por sua vez, acontecer com alguma regularidade em atividades relacionadas à codificação de sistemas. A adoção de padrões representa, em termos gerais, uma forma de reuso de soluções de software cuja validade já foi testada e comprovada ao longo de diversos projetos. A partir da utilização de patterns busca-se a obtenção de aplicações melhor estruturadas, assim como mais flexíveis diante de mudanças que serão comuns ao longo do ciclo de vida de um software.

Por estar focado normalmente na solução de uma questão pontual, um padrão de projeto não está necessariamente ligado ao uso de uma tecnologia específica. É bastante comum que um mesmo design pattern possa ser empregado sem maiores dificuldades nas mais diferentes plataformas de desenvolvimento (Delphi, C#, VB.NET, Java, C++, etc.).

# tory, Observer e Iterator

Renato José Groffe

Os primeiros esforços na definição de padrões dentro da área de programação aconteceram ainda na década de 1990, tendo resultado na publicação de 23 patterns conhecidos como GoF. Desde então, diversos outros padrões viriam a ser catalogados, oferecendo assim um amplo leque de soluções aos desenvolvedores de software.

### Nota do DevMan

GoF é a sigla em inglês para "Gang of Four", sendo um apelido dado ao trabalho conjunto de 4 autores (Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides), os quais documentaram 23 padrões no famoso livro "Design Patterns: Elements of Reusable Object-Oriented Software". Tais patterns foram especificados na época tomando-se como base exemplos elaborados em C++ e Smalltalk; no entanto, a ampla abrangência destes padrões possibilitou que os mesmos sejam utilizados ainda hoje, auxiliando desenvolvedores Delphi, .NET e Java na resolução de uma série de situações recorrentes no cotidiano.

Considerando o tipo de problema a que se propõem resolver, os patterns GoF podem ser divididos em:

- Padrões de criação: disponibilizam técnicas para a geração de novas instâncias de objetos;
- Padrões estruturais: definem formas para se trabalhar com a composição de objetos;
- Padrões comportamentais: estabelecem regras para a interação entre classes ou, ainda, definem como serão as responsabilidades das mesmas.

Convém destacar ainda o fato de que muitos patterns costumam ser representados graficamente através da noção UML, sobretudo através de diagrama de classes que demonstram os diversos elementos que definem um padrão.

### Nota do DevMan

UML (sigla em inglês para "Unified Modeling Language") é uma linguagem voltada à modelagem de componentes de software e que se baseia em conceitos de orientação a objetos. Trata-se de uma notação que disponibiliza uma série de diagramas, sendo o de classes um dos mais conhecidos (com este último sendo utilizado para a demonstração dos diversos patterns descritos neste artigo).

## Criando famílias de objetos com o padrão Abstract Factory

Através do padrão de criação Abstract Factory (**Figura 1**) é definido o uso de uma interface (ou classe abstrata) que contém a estrutura para a geração de famílias de objetos relacionados (ou mesmo dependentes entre si), sem que contudo se especifiquem as classes concretas das referências que serão criadas.

Diante disso, o consumidor de uma Abstract Factory não precisará possuir a noção de todas as classes concretas que implementam a mesma ou, mesmo, dos tipos que correspondem aos produtos devolvidos pelas operações de tal Factory.

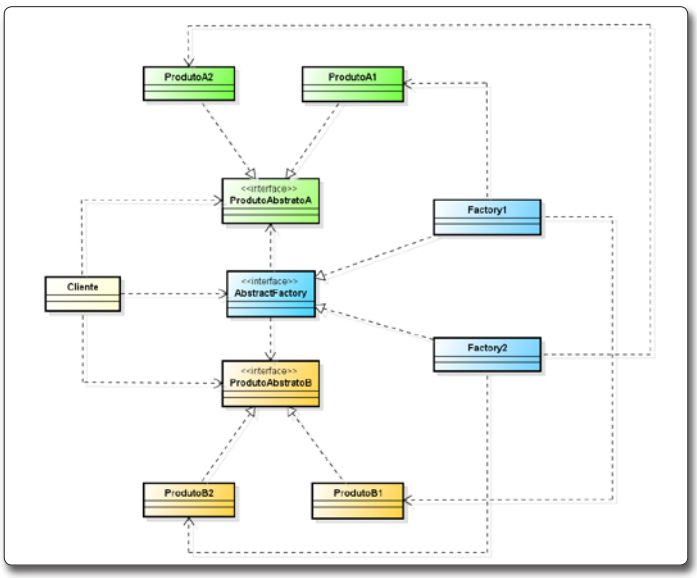



**Figura 1.** Representação esquemática do pattern Abstract Factory

As instâncias correspondentes às fabricas de objetos e aos produtos destas serão obtidas, basicamente, através de implementações concretas de interfaces e classes abstratas: a forma de se determinar quais elementos servirão de base para a geração de referências dependerá, geralmente, de informações de configuração ou, mesmo, de algum tipo de parâmetro fornecido por um usuário dentro de uma aplicação.

**Nota**

Os diagramas de classe exibidos neste artigo foram criados a partir da aplicação Astah UML. Esta ferramenta disponibiliza tanto versões gratuitas quanto proprietárias, sendo que na seção de Links consta o endereço a partir do qual o download da mesma poderá ser realizado.

## Acessando objetos seqüencialmente com o padrão Iterator

O padrão comportamental conhecido como Iterator (**Figura 2**) serve de base para a criação de mecanismos de acesso sequencial a um agrupamento de objetos. A partir disto, estruturas que dependam

de uma coleção de referências não precisarão ter o conhecimento de como estes elementos encontram-se organizados, já que a implementação deste pattern fornece todos os recursos necessários para a navegação ao longo de um conjunto de objetos.

### Nota do DevMan

Acoplamento é um conceito computacional que indica o grau de relacionamento entre diferentes partes de uma aplicação. Um alto acoplamento implica em um nível de alta dependência entre os diversos componentes envolvidos, fato este que pode resultar em dificuldades na manutenção futura das funcionalidades envolvidas. Logo, o ideal é que exista um baixo nível de acoplamento entre as estruturas que constituem um determinado elemento.

O acoplamento está ligado a outra noção de grande relevância na área de software: a coesão. Esta última deve ser compreendida como a medida com a qual um item (classe ou componente) atende ao objetivo inicial que justificou a sua construção. Uma alta coesão indica que o item que se está considerando não acumula responsabilidades além daquelas para as quais foi especificado, sendo esta uma característica desejável para a grande maioria dos projetos de software.

Normalmente, um baixo acoplamento será consequência direta de um alto grau de coesão ao se tomarem por base os componentes de um sistema.

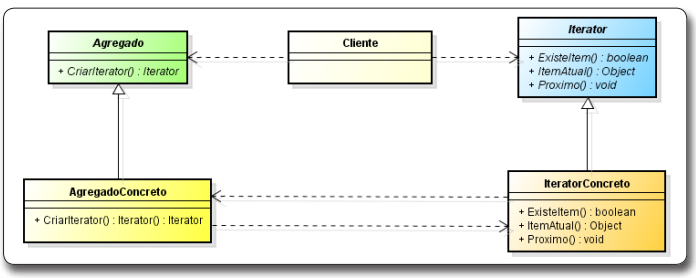



**Figura 2.** Representação esquemática do pattern Iterator

Iterator está fundamentado em um conceito chamado de Princípio da Responsabilidade Única: tipos concretos que definam coleções de objetos não implementam as funções de navegação pelos mesmos; isto será atribuição de outras classes que estarão relacionadas a estes tipos de estruturas e que implementarão tais comportamentos.

A implementação do pattern Iterator pode ser feita a partir dos seguintes elementos básicos:

- Uma classe que determina como acontece a iteração (tipo abstrato Iterator);
- A especificação referente a uma coleção de objetos (tipo abstrato Agregado).

As classes concretas do agregado e do iterador relacionam-se, sendo que a partir da primeira estrutura se obtém o elemento para navegação, ao passo que este último terá conhecimento de como o conjunto de objetos a ser manipulado encontra-se disposto.

## Observer

O padrão comportamental Observer (**Figura 3**) permite que se estabeleça um relacionamento de um-para-muitos entre tipos de objetos diferentes, de maneira que modificações em um estado vinculado a uma instância principal desencadeie uma ação específica nas referências que dependam da mesma. Este pattern também é conhecido por outros nomes como Dependents, Event Generator e Publish-Subscribe.

A utilização do padrão Observer envolve os seguintes aspectos:

- A especificação de uma classe abstrata ou interface (Observable), com esta estrutura sendo o elemento a partir do qual se originam as notificações enviadas a objetos que dependam da mesma;
- Um tipo básico (Observer), a partir do qual serão recebidas mensagens que indicam uma atualização no status daquilo que se está observando (Observable).

### Nota do DevMan

O Princípio da Responsabilidade Única (tradução do inglês "Single Responsibility Principle") tem por idéia central a proposta de limitar cada classe de um projeto a somente um tipo de responsabilidade. Levando tal aspecto em consideração, haverá somente uma razão para se modificar uma determinada classe, sendo que isto deve acontecer de forma que se evitem efeitos colaterais em outras estruturas que dependam do tipo a ser alterado. A aplicação deste conceito busca a obtenção de componentes mais coesos dentro de uma solução de software.

Classes com um número extenso de funcionalidades e, portanto, caracterizadas por uma baixa coesão, estão na contramão do que prega o Princípio da Responsabilidade Única: modificações que parecem simples num primeiro momento podem resultar em erro ao longo de todo um sistema.

Considerando as classes que precisarão ser implementadas quando da utilização deste padrão, o tipo a ser observado poderá contar com operações para que observadores possam se registrar junto ao mesmo (ou não), além de um método que dispare o envio de notificações a estes últimos na eventualidade de uma alteração de status. Já em relação a tipos que definam o observador, o comum neste caso será a existência de um método através do qual serão recebidas as atualizações indicando modificações na estrutura observada.

### Nota do DevMan

Importante ressaltar ainda o fato de que o padrão Observer serviu de base para a implementação de outra idéia presente nas linguagens modernas: a programação orientada a eventos. Dentro deste paradigma um evento corresponde ao tratamento que se faz em reposta a alguma ação desencadeada por um usuário ou, até mesmo, como consequência do processamento interno de uma determinada operação em uma aplicação.

## Exemplos de uso dos patterns Abstract Factory, Iterator e Observer

O desenvolvimento das aplicações demonstradas neste artigo foi realizado por meio da utilização do Embarcadero Delphi XE2, sendo que através das mesmas serão apresentadas formas de uso dos patterns Abstract Factory, Iterator e Observer.

## Utilizando o pattern Abstract Factory

A fim de se construir uma aplicação de exemplo que empregue o padrão Abstract Factory, o primeiro passo será criar um projeto do tipo "VCL Forms Application" chamado TesteAbstractFactory, sendo que o mesmo envolve, de maneira simplificada, a obtenção

de informações sobre o mecanismo empregado para a geração de uma forma energia, bem como o recurso que serve de base neste processo. Estes dois tipos de características correspondem, basicamente, aos produtos a serem retornados por meio das classes que implementam a interface da Abstract Factory apresentada mais adiante.

## Criação das classes que empregam o padrão Abstract Factory

Na **Listagem 1** são demonstradas as especificações das interfaces IUsinaEnergiaFactory, IMecanismoGeracao e IRecursoProducaoEnergia. IUsinaEnergiaFactory corresponde à interface da Abstract Factory que estará sendo gerada neste exemplo. Já IMecanismoGeracao (informações sobre o dispositivo empregado na produção de energia) e IRecursoProducaoEnergia() representam as partes que constituem a família de produtos retornados pelas implementações da Factory aqui descrita.

**Listagem 1**. Interfaces IMecanismoGeracao, IRecursoProducaoEnergia e IUsinaEnergiaFactory

```
unit InterfacesEnergia;

interface

type
 IMecanismoGeracao = interface
  function ObterInformacoesMecanismo: string;
 end;

 IRecursoProducaoEnergia = interface
  function ObterInformacoesRecurso: string;
 end;

 IUsinaEnergiaFactory = interface
  function ObterMecanismoGeracao:
  IMecanismoGeracao;
  function ObterRecursoProducaoEnergia:
  IRecursoProducaoEnergia;
 end;

implementation

end.
```

A **Listagem 2** contém as definições dos tipos responsáveis pelo fornecimento de informações sobre a produção de energia atômica. TUsinaEnergiaAtomicaFactory representa a implementação concreta da Abstract Factory definida anteriormente. Já TMecanismoGeracaoEnergiaAtomica e TRecursoProducaoEnergiaAtomica servirão de base para a geração das instâncias a serem devolvidas por esta Factory.

Já a **Listagem 3** exibe o código-fonte da Factory TUsinaEnergiaEolicaFactory, com esta última sendo utilizada para o acesso a informações sobre a geração de energia eólica. TMecanismoGeracaoEnergiaAtomica e TRecursoProducaoEnergiaAtomica são as classes concretas das referências que serão retornadas por TUsinaEnergiaAtomicaFactory.

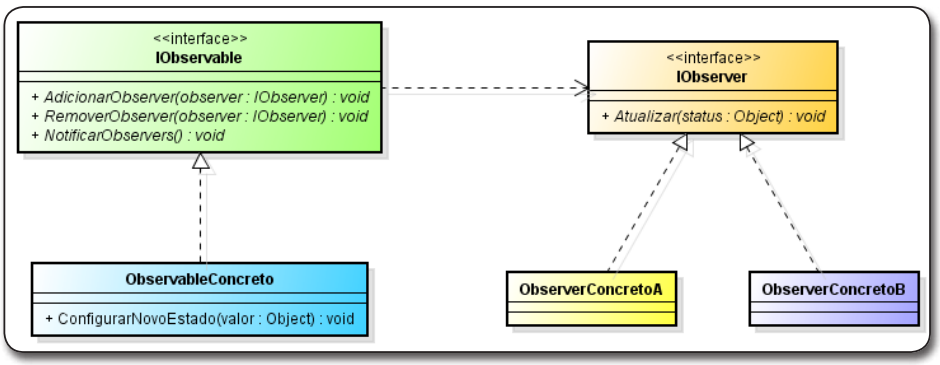



**Figura 3.** Representação esquemática do pattern Observer

**Listagem 2**. Classes TMecanismoGeracaoEnergiaAtomica, TRecursoProducaoEnergiaAtomica e TUsinaEnergiaAtomicaFactory

```
type
 TMecanismoGeracaoEnergiaAtomica = class(TInterfacedObject, IMecanismoGeracao)
 public
  function ObterInformacoesMecanismo: string;
 end;
 TRecursoProducaoEnergiaAtomica = class(TInterfacedObject, IRecursoProducaoEnergia)
 public
  function ObterInformacoesRecurso: string;
 end;

 TUsinaEnergiaAtomicaFactory = class(TInterfacedObject, IUsinaEnergiaFactory)
 public
  function ObterMecanismoGeracao: IMecanismoGeracao;
  function ObterRecursoProducaoEnergia: IRecursoProducaoEnergia;
 end;

implementation


 { TMecanismoGeracaoEnergiaAtomica }

 function TMecanismoGeracaoEnergiaAtomica.ObterInformacoesMecanismo: string;
 begin
  Result:= 'Reator de Fissão Nuclear';
 end;
 { TRecursoProducaoEnergiaAtomica }
 function TRecursoProducaoEnergiaAtomica.ObterInformacoesRecurso: string;
 begin
  Result:= 'Plutônio-238';
 end;
 { TUsinaEnergiaAtomicaFactory }
 function TUsinaEnergiaAtomicaFactory.ObterMecanismoGeracao: IMecanismoGeracao;
 begin
  Result:= TMecanismoGeracaoEnergiaAtomica.Create;
 end;
 function TUsinaEnergiaAtomicaFactory.ObterRecursoProducaoEnergia: IRecursoProducaoEnergia;
 begin
  Result:= TRecursoProducaoEnergiaAtomica.Create;
 end;

end.
```

**Listagem 3**. Classes TMecanismoGeracaoEnergiaEolica, TRecursoProducaoEnergiaEolica e TUsinaEnergiaEolicaFactory

```
type
 TMecanismoGeracaoEnergiaEolica = class(TInterfacedObject, IMecanismoGeracao)
 public
  function ObterInformacoesMecanismo: string;
 end;

 TRecursoProducaoEnergiaEolica = class(TInterfacedObject, IRecursoProducaoEnergia)
 public
  function ObterInformacoesRecurso: string;
 end;

 TUsinaEnergiaEolicaFactory = class(TInterfacedObject, IUsinaEnergiaFactory)
 public
  function ObterMecanismoGeracao: IMecanismoGeracao;
  function ObterRecursoProducaoEnergia: IRecursoProducaoEnergia;
 end;

implementation

 { TMecanismoGeracaoEnergiaEolica }

 function TMecanismoGeracaoEnergiaEolica.ObterInformacoesMecanismo: string;
 begin
  Result:= 'Turbina Eólica';
 end;

 { TRecursoProducaoEnergiaEolica }

 function TRecursoProducaoEnergiaEolica.ObterInformacoesRecurso: string;
 begin
  Result:= 'Ventos Litorâneos';
 end;

 { TUsinaEnergiaEolicaFactory }

 function TUsinaEnergiaEolicaFactory.ObterMecanismoGeracao: IMecanismoGeracao;
 begin
  Result:= TMecanismoGeracaoEnergiaEolica.Create;
 end;

 function TUsinaEnergiaEolicaFactory.ObterRecursoProducaoEnergia:
 IRecursoProducaoEnergia;
 begin
  Result:= TRecursoProducaoEnergiaEolica.Create;
 end;

end.
```

## Implementação da tela para testes do padrão Abstract Factory

Uma vez que o processo de codificação das classes relativas ao fornecimento de informações sobre produção de energia tenha finalizado, deve-se prosseguir com a implementação da tela que será empregada para os testes com as mesmas. O formulário principal da aplicação deve ser renomeado para FormTeste. Na **Listagem 4** está sendo apresentada a implementação do mesmo.

O método btnObterInformacoesClick corresponde ao evento OnClick do botão btnObterInformacoes. A partir do mesmo será utilizada uma implementação concreta da interface IUsinaEnergiaFactory (TUsinaEnergiaAtomicaFactory ou TUsinaEnergiaEolicaFactory), acionando-se em seguida o método ExibirInformacoesTipoEnegia.

**Listagem 4**. Implementação do formulário TFormTeste

```
procedure TFormTeste.ExibirInformacoesTipoEnegia(
 factory: IUsinaEnergiaFactory);
var
 mecanismo: IMecanismoGeracao;
 recurso: IRecursoProducaoEnergia;
begin
 mecanismo:= factory.ObterMecanismoGeracao;
 recurso:= factory.ObterRecursoProducaoEnergia;

 ShowMessage('Mecanismo de geração de energia: ' +
       mecanismo.ObterInformacoesMecanismo + #13#10#13#10 +
       'Recurso utilizado para produção: ' +
       recurso.ObterInformacoesRecurso);
end;

procedure TFormTeste.btnObterInformacoesClick(Sender: TObject);
var
 factory: IUsinaEnergiaFactory;
begin
 if (rdbEnergiaAtomica.Checked) then
  factory:= TUsinaEnergiaAtomicaFactory.Create
 else
  factory:= TUsinaEnergiaEolicaFactory.Create;

 ExibirInformacoesTipoEnegia(factory);
end;
```

A operação recebe como parâmetro uma Factory baseada no tipo IUsinaEnergiaFactory. Através deste último serão obtidas instâncias IMecanismoGeracao e IRecursoProducaoEnergia, com estes objetos sendo empregados na exibição de informações sobre o mecanismo e a fonte de origem de uma determinada forma de energia.

## Executando a aplicação para testes do padrão Abstract Factory

Ao se executar o projeto TesteAbstractFactory será exibida uma tela como a indicada na **Figura 4**, na qual deverá ser selecionada alguma das opções para o teste do padrão Abstract Factory (Energia Atômica ou Eólica).

Caso seja selecionada a opção "Energia Atômica", o usuário irá se deparar com a janela apresentada na **Figura 5**.

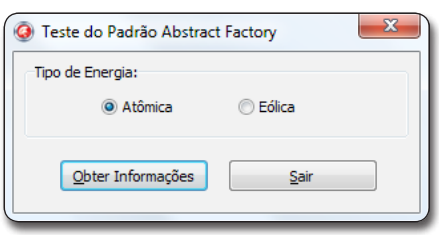



**Figura 4.** Tela inicial que é exibida ao se executar o projeto TesteAbstractFactory

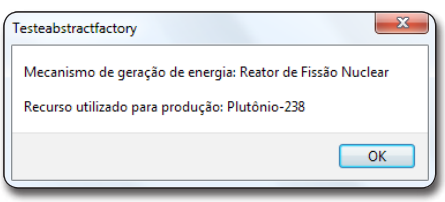



**Figura 5.** Informações sobre Energia Atômica

## Utilizando o pattern Iterator

Para se testar o uso do Iterator deverá ser criado um projeto do tipo "VCL Forms Application" chamado TesteIterator. A partir do executável resultante serão exibidas em tela informações (ano do evento e vencedor) sobre as diferentes Copas do Mundo já disputadas.

## Criação das classes que empregam o padrão Iterator

Na **Listagem 5** está definida a interface IIterator, a qual determina a estrutura a ser implementada por classes que disponibilizem funcionalidades navegação seqüencial por coleções de objetos. Já **Listagem 6** apresenta o código fonte da interface IColecaoSequencial, a qual corresponde à estrutura para a manipulação de agrupamentos de objetos por meio de um "iterador" (operação CriarIterator).

**Listagem 5**. Interface IIterator

```
unit Iterator;

interface

type
 IIterator = interface
  function ExisteItem: Boolean;
  function ItemAtual: TObject;
  procedure MoverParaProximo;
 end;

implementation

end.
```

**Listagem 6**. Interface IColecaoSequencial

```
unit ColecaoSequencial;

interface

uses Iterator;

type
 IColecaoSequencial = interface
  function CriarIterator: IIterator;
 end;

implementation

end.
```

O tipo TIteratorListaSequencial (**Listagem 7**) representa uma implementação concreta da interface IIterator: é por meio desta classe que as diversas operações que caracterizam a navegação ao longo de uma coleção de objetos serão executadas (checagem de existência de itens para determinar se foi atingido ou não o fim da seqüência, obtenção da referência que corresponde à posição atual do iterador, deslocamento para o próximo item do agrupamento de objetos).TIteratorListaSequencial está recebendo em seu construtor um objeto do tipo TList, com este último correspondendo à coleção de objetos em que ocorrerão as atividades de iteração.

A classe TListaSequencial (**Listagem 8**) implementa a interface IColecaoSequencial, contando com métodos para adicionar novos itens a uma coleção de objetos (operação AdicionarItem) e obtenção de um iterator para a navegação sequencial ao longo de tal agrupamento (operação CriarIterator, com esta última sendo gerada com base na referência ao tipo TList e que é representada pelo atributo listaSequencial).

**Listagem 7**. Classe TIteratorListaSequencial

```
type
 TIteratorListaSequencial = class(TInterfacedObject, IIterator)
 private
  listaSequencial: TList;
  posicaoAtual: Integer;
 public
  constructor Create(listaBase: TList);
  function ExisteItem: Boolean;
  function ItemAtual: TObject;
  procedure MoverParaProximo;
 end;

implementation

 { TIteratorListaSequencial }

 constructor TIteratorListaSequencial.Create(listaBase: TList);
 begin
  listaSequencial:= listaBase;
  posicaoAtual:= 0;
 end;

 function TIteratorListaSequencial.ExisteItem: Boolean;
 begin
  if (posicaoAtual < listaSequencial.Count) then
   Result:= True
  else
   Result:= False;
 end;

 function TIteratorListaSequencial.ItemAtual: TObject;
 begin
  Result:= listaSequencial.Items[posicaoAtual];
 end;
 procedure TIteratorListaSequencial.MoverParaProximo;
 begin
  posicaoAtual:= posicaoAtual + 1;
 end;

end.
```

**Listagem 8**. Classe TListaSequencial

```
constructor TListaSequencial.Create;
begin
 listaSequencial:= TList.Create;
end;

function TListaSequencial.CriarIterator: IIterator;
begin
 Result:= TIteratorListaSequencial.Create(listaSequencial);
end;

procedure TListaSequencial.AdicionarItem(objeto: TObject);
begin
 listaSequencial.Add(objeto);
end;
```

Finalmente a **Listagem 9** apresenta a definição da classe TCopa. Este tipo será empregado para a exibição de informações sobre Copas do Mundo (ano de realização, equipe campeã), sendo que isto acontecerá por meio da iteração em uma coleção de objetos que fará uso desta estrutura.

**Listagem 9**. Classe TCopa

```
type
 TCopa = class
 private
  fAno: Integer;
  fCampeao: string;
 public
  property Ano: Integer read fAno write fAno;
  property Campeao: string read fCampeao write fCampeao;
 end;
```

## Implementação da tela para testes do padrão Iterator

Assim que se concluírem as atividades de codificação das estruturas envolvidas no uso do pattern Iterator, chega o momento de se implementar a tela a partir da qual serão efetuados os testes. O formulário principal da aplicação terá sua propriedade Name configurada com o valor "FormTeste". A **Listagem 10** apresenta a implementação desta tela.

**Listagem 10**. Implementação do formulário TFormTeste

```
procedure TFormTeste.NavegarPorIterator(objetoIteracao: IIterator);
begin
 while (objetoIteracao.ExisteItem) do
 begin
  ExibirInformacoesItemAtual(objetoIteracao.ItemAtual);
  objetoIteracao.MoverParaProximo;
 end;
end;

procedure TFormTeste.ExibirInformacoesItemAtual(itemAtual: TObject);
var
 informacoesCopa: TCopa;
begin
 if (itemAtual is TCopa) then
 begin
  informacoesCopa:= (itemAtual as TCopa);
  ShowMessage(
   'Ano da Copa: ' + IntToStr(informacoesCopa.Ano) +
   #13#10#13#10 +
   'Campeão: ' + informacoesCopa.Campeao);
 end
 else
 begin
  ShowMessage('O item atual não corresponde a um objeto do ' +
   'tipo TCopa.');
 end;
end;

procedure TFormTeste.btnObterInformacoesClick(Sender: TObject);
var
 listaSequencial: IColecaoSequencial;
begin
 listaSequencial:= ObterListaCopasFutebol;
 NavegarPorIterator(listaSequencial.CriarIterator);
end;
```

O método btnObterInformacoesClick define as ações a serem disparadas para o evento OnClick do botão btnObterInformacoes. Através desta operação a função ObterListaCopasFutebol é invocada, obtendo-se assim uma coleção de objetos com informações de todos os torneios disputados; a chamada ao método NavegarPorIterator fará uso do iterator para a navegação seqüencial pelo agrupamento de objetos do tipo TCopa, exibindo-se por meio da chamada ao procedimento ExibirInformacoesItemAtual os dados contidos em cada referência encontrada.

## Executando a aplicação para testes do padrão Iterator

Ao se disparar a execução do projeto TesteIterator será exibida inicialmente a tela apresentada na **Figura 6**. Será necessário, a partir disto, clicar na opção "Obter Informações" a fim de serem exibidas informações sobre diferentes Copas (um exemplo está na **Figura 7**).

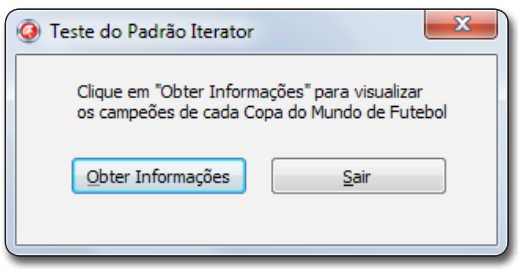



**Figura 6.** Tela inicial que é exibida ao se executar o projeto TesteIterator

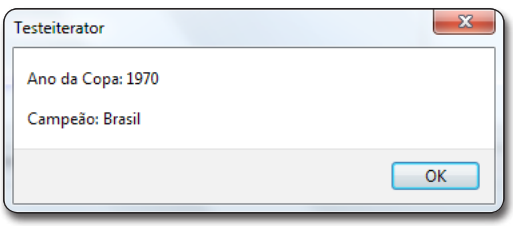



**Figura 7.** Informações de objeto obtido por meio de um Iterator

## Utilizando o pattern Observer

Para construir uma aplicação de exemplo que empregue o padrão Observer o primeiro passo será criar um projeto do tipo "VCL Forms Application" chamado TesteObserver, sendo que o mesmo envolve, de maneira simplificada, a atualização do valor de uma ação negociada em bolsa de valores e a notificação de tal ocorrência a corretoras que possam estar interessadas em tal fato.

## Criação das classes que empregam o padrão Observer

Na **Listagem 11** são apresentadas as definições para as classes abstratas TObserverAcoes e TAcaoObservable. O tipo TObserverAcoes corresponde à estrutura que receberá notificações quando da ocorrência de atualizações de status em TAcao Observable (via método Atualizar). Esta última classe contém operações para a adição / remoção de observadores (métodos AdicionarObserver e RemoverObserver, respectivamente) e o envio de mensagens de aviso para estes elementos (método NotificarObservers).

**Listagem 11**. Classes abstratas TObserverAcoes e TAcaoObservable

```
type
 TObserverAcoes = class
 public
  procedure Atualizar(codigoAcao: string;
   precoAcao: Real); virtual; abstract;
 end;

 TAcaoObservable = class
  procedure AdicionarObserver(observer: TObserverAcoes);
   virtual; abstract;
  procedure RemoverObserver(observer: TObserverAcoes);
   virtual; abstract;
  procedure NotificarObservers(); virtual; abstract;
 end;
```

Já na **Listagem 12** é demonstrada a definição das classes TCorretoraValoresA e TCorretoraValoresB, com ambas funcionando como observadores já que implementação o tipo abstrato TObserverAcoes. O método Atualizar presente nestas duas estruturas será a base para que um objeto baseado no tipo TAcaoObservable envie notificações de alteração de status.

**Listagem 12**. Classes TCorretoraValoresA e TCorretoraValoresB

```
type
 TCorretoraValoresA = class(TObserverAcoes)
  procedure Atualizar(codigoAcao: string; precoAcao: Real); override;
 end;

 TCorretoraValoresB = class(TObserverAcoes)
  procedure Atualizar(codigoAcao: string; precoAcao: Real); override;
 end;

implementation

{ TCorretoraValoresA }

procedure TCorretoraValoresA.Atualizar(codigoAcao: string;
 precoAcao: Real);
begin
 ShowMessage('Corretora A - Atualização no valor de Ação' +
  #13#10#13#10 + 'Código do Título:' + codigoAcao +
  #13#10#13#10 + 'Valor Atual de Negociação:' +
  FormatFloat('0.00', precoAcao));
end;

{ TCorretoraValoresB }

procedure TCorretoraValoresB.Atualizar(codigoAcao: string;
 precoAcao: Real);
begin
 ShowMessage('Corretora B - Atualização no valor de Ação' +
  #13#10#13#10 + 'Código do Título:' + codigoAcao +
  #13#10#13#10 + 'Valor Atual de Negociação:' +
  FormatFloat('0.00', precoAcao));
end;
```

O tipo TAcao (**Listagem 13**) implementa a classe abstrata TAcao Observable, representando uma ação que é negociada na Bolsa de Valores. Os métodos AdicionarObserver, RemoverObserver e NotificarObservers correspondem às funcionalidades que determinam as características de uma estrutura baseada no padrão Observer (registro / desvinculação de um observador, notificações de todos os observadores associados ao item observável).

Já a alteração do preço de um título é feita invocando-se o método AlterarPreco: todas as corretoras associadas à lista "listaObservers" serão avisadas desta mudança, acionando-se para isto a operação Atualizar de cada instância baseada em TObserverAcoes.

## Implementação da tela para testes do padrão Observer

Encerrada a codificação das classes que fazem uso do padrão Observer, deve-se prosseguir com a implementação da tela que será empregada no teste das mesmas. O formulário principal da aplicação deve ser renomeado para FormTeste. Na **Listagem 13** está sendo apresentada a implementação desta janela.

**Listagem 13**. Classe TAcao

```
type
 TAcao = class(TAcaoObservable)
 private
  codigoAcao: string;
  precoAcao: Real;
  listaObservers: TList;
 public
  constructor Create(codAcao: string; preco: Real);
  procedure AlterarPreco(preco: Real);
  procedure AdicionarObserver(observer: TObserverAcoes); override;
  procedure RemoverObserver(observer: TObserverAcoes); override;
  procedure NotificarObservers();
 end;
implementation
{ TAcao }
constructor TAcao.Create(codAcao: string; preco: Real);
begin
 codigoAcao:= codAcao;
 precoAcao:= preco;
 listaObservers:= TList.Create;
end;

procedure TAcao.AlterarPreco(preco: Real);
begin
 precoAcao:= preco;
 NotificarObservers;
end;

procedure TAcao.AdicionarObserver(observer: TObserverAcoes);
begin
 listaObservers.Add(observer);
end;

procedure TAcao.RemoverObserver(observer: TObserverAcoes);
begin
 listaObservers.Remove(observer);
end;
procedure TAcao.NotificarObservers;
var
  i: Integer;
  objObserver: TObject;
  observer: TObserverAcoes;
begin
 for i := 0 to listaObservers.Count - 1 do
 begin
  objObserver:= listaObservers.Items[i];
  observer:= (objObserver as TObserverAcoes);
  observer.Atualizar(codigoAcao, precoAcao);
 end;
end;
```

No evento FormCreate é gerada uma instância baseada no tipo TAcao, além de referência para as corretoras que são representadas pelas classes TCorretoraValoresA e TCorretoraValoresB. Os eventos chkCorretoraAClick e chkCorretoraBClick são executados como resposta a um clique nos controles chkCorretoraA e chkCorretoraB, respectivamente: caso um CheckBox esteja ativo, o observador associado ao mesmo será vinculado à instância de nome "acaoXYZ"; do contrário, desvincula-se a referência que equivale a uma corretora, de forma que a mesma não receba mais notificações de alteração no preço de um título.

O método btnAtualizarPrecoClick corresponde ao evento OnClick do botão btnAtualizarPreco. Quando esta operação for disparada será atribuído um novo preço à instância do tipo TAcao empregada para testes, com possíveis observadores vinculados à mesma sendo notificados disto (e exibindo como resultado mensagens em tela).

**Listagem 13**. Implementação do formulário TFormTeste

```
procedure TFormTeste.FormCreate(Sender: TObject);
begin
  acaoXYZ:= TAcao.Create('XYZ', 100);
  txtPrecoAcao.Text:= '100,00';
  corretoraA:= TCorretoraValoresA.Create;
  corretoraB:= TCorretoraValoresB.Create;
end;
procedure TFormTeste.chkCorretoraAClick(Sender: TObject);
begin
  if (chkCorretoraA.Checked) then
    acaoXYZ.AdicionarObserver(corretoraA)
  else
    acaoXYZ.RemoverObserver(corretoraA);
end;
procedure TFormTeste.chkCorretoraBClick(Sender: TObject);
begin
  if (chkCorretoraB.Checked) then
    acaoXYZ.AdicionarObserver(corretoraB)
  else
    acaoXYZ.RemoverObserver(corretoraB);
end;
procedure TFormTeste.btnAtualizarPrecoClick(Sender: TObject);
var
    preco: Real;
begin
  try
    preco:= StrToFloat(txtPrecoAcao.Text);
    (acaoXYZ as TAcao).AlterarPreco(preco);
  except
    ShowMessage('Erro na atualização do Preço da Ação!');
  end;
end;
```

## Executando a aplicação para testes do padrão Observer

Ao se executar o projeto TesteObserver será exibida uma tela como a indicada na **Figura 8**, na qual deverão estar sendo selecionadas as operadoras a serem notificadas a respeito de alterações de preço no valor de ações de empresa XYZ.

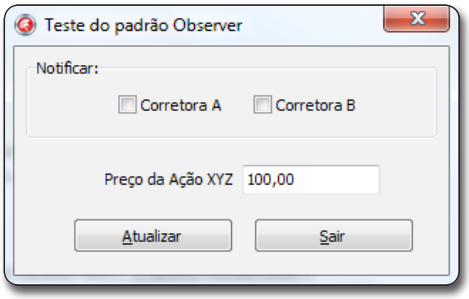



**Figura 8.** Tela inicial que é exibida ao se executar o projeto TesteObserver

Para testar o funcionamento do pattern Observer marcam-se as corretoras A e B, além de se alterar o valor da ação XYZ; clica-se na sequência no botão "Atualizar": serão exibidas então as mensagens de atualização das estruturas observadoras referentes às empresas A e B.

## Conclusão

Conforme demonstrado no transcorrer deste artigo, design patterns são técnicas que podem ser utilizadas na solução de questões específicas dentro do desenvolvimento de sistemas. Diversos padrões foram estabelecidos ao longo da evolução da área de software, sendo que a escolha da melhor opção para cada caso dependerá sempre dos requisitos e das características do projeto que se está considerando.

**Renato José Groffe**
*renato.groffe@yahoo.com.br*
Atua como consultor em atividades voltadas ao desenvolvimento de softwares há mais de 10 anos. Bacharel em Sistemas de Informação, com especialização em Engenharia de Software. Microsoft Certified Technology Specialist (Web, WCF, Distributed Applications, ADO.NET, Windows Forms), Oracle Certified Associate (PL/SQL), Sun Certified (SCJP, SCWCD), ITIL Foundation V2.

**Unified Modeling Language**
*http://www.uml.org/*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

## A DevMedia simplifica pra você!

easy .net magazine

### Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

### Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

**A Easy.Net não é feita em PDF ou Flash!**
**Ela utiliza um aplicativo 100% online.**

DevMedia group

**Acesse agora a edição Nº 1 totalmente grátis e comprove**

**www.devmedia.com.br/easynet**

**Easy.Net Magazine, seu curso inicial de .Net!**

# Conhecimento faz diferença!

**Faça já sua assinatura digital ! | www.devmedia.com.br/es**

## Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia